1990-06

Error Control Coding for Multi-Frequency Modulation

Ives, Robert W.

Monterey, California: Naval Postgraduate School

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A232 421

THESIS

ERROR CONTROL CODING FOR
MULTI-FREQUENCY MODULATION

by

Robert W. Ives

June 1990

Thesis Advisor: P. H. Moose

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (if applicable): 32	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (city, state, and ZIP code): Monterey, CA 93943-5000	7b Address (city, state, and ZIP code): Monterey, CA 93943-5000
8a Name of Funding Sponsoring Organization; 8b Office Symbol (if applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers: Program Element No; Project No; Task No; Work Unit Accession No

11 Title (include security classification): ERROR CONTROL CODING FOR MULTI-FREQUENCY MODULATION

12 Personal Author(s): Robert W. Ives

13a Type of Report	13b Time Covered	14 Date of Report (year, month, day)	15 Page Count
Master's Thesis	From To	June 1990	81

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number)
Field	Group	Subgroup	Multi-Frequency Modulation, Reed-Solomon, Error Control

19 Abstract (continue on reverse if necessary and identify by block number)

Multi-frequency modulation (MFM) has been developed at NPS using both quadrature-phase-shift-keyed (QPSK) and quadrature-amplitude-modulated (QAM) signals with good bit error performance at reasonable signal-to-noise ratios. Improved performance can be achieved by the introduction of error control coding.

This report documents a Fortran simulation of the implementation of error control coding into an MFM communication link with additive white Gaussian noise. Four Reed-Solomon codes were incorporated, two for 16-QAM and two for 32-QAM modulation schemes. The error control codes used were modified from the conventional Reed-Solomon codes in that one information symbol was sacrificed to parity in order to use a simplified decoding algorithm which requires no iteration and enhances error detection capability. Bit error rates as a function of SNR and E_b/N_0 were analyzed, and bit error performance was weighed against reduction in information rate to determine the value of the codes.

20 Distribution Availability of Abstract	21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified	
22a Name of Responsible Individual: P.H. Moose	22b Telephone (include Area code): (408) 646-2838	22c Office Symbol: 62ME

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete

Approved for public release; distribution is unlimited.

Error Control Coding
for
Multi-Frequency Modulation

by

Robert W. Ives
Lieutenant, United States Navy
B.S., U.S. Naval Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: 

Robert W. Ives

Approved by: 

P.H. Moose, Thesis Advisor

T.A. Schwendter, Second Reader

J.P. Powers, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

Multi-frequency modulation (MFM) has been developed at NPS using both quadrature-phase-shift-keyed (QPSK) and quadrature-amplitude-modulated (QAM) signals with good bit error performance at reasonable signal-to-noise ratios. Improved performance can be achieved by the introduction of error control coding.

This report documents a Fortran simulation of the implementation of error control coding into an MFM communication link with additive white Gaussian noise. Four Reed-Solomon codes were incorporated, two for 16-QAM and two for 32-QAM modulation schemes. The error control codes used were modified from the conventional Reed-Solomon codes in that one information symbol was sacrificed to parity in order to use a simplified decoding algorithm which requires no iteration and enhances error detection capability. Bit error rates as a function of SNR and E_b/N_0 were analyzed, and bit error performance was weighed against reduction in information rate to determine the value of the codes.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Many modern communication methods which merge communications and computer technology require efficient and reliable transfer of digital data from an information source to a destination. This requirement is not always an easy one to achieve. Because this transmission is through a physical medium which is subject to various types of noise, distortion and interference, the data output from the channel may differ from its input. The demand for reliable communication necessitates the detection and possibly the correction of those errors resulting from impaired transmission. These errors can be detected and corrected using the principles of error control coding.

Error control coding can be implemented in a communication system usually at the cost of data transmission rate. Generally, the code that corrects more errors will require more parity symbols, reducing the data rate. On the other hand, a code which corrects more errors will be more reliable in a high noise environment. The balance between data transmission rate and coding gain poses an interesting problem, one which will be addressed in this thesis.

The subject of this thesis is the simulation and analysis of various error control codes in a bandpass Multi-Frequency Modulation (MFM) communication link with Additive White Gaussian Noise (AWGN). MFM is a signal modulation format which is readily adaptable to many digital communication link scenarios. Its efficient use of the frequency spectrum allows very high data transmission rates. But like all communication systems, MFM is subject to errors.

There are four error control codes used in the simulation, all of which are Reed-Solomon. Two error correction detection capability codes (single error correcting double error detecting (SEC DED) and double error correcting triple error detecting (DEC TED)) in combination with three different modulation schemes (one 16-QAM and two 32-QAM constellations) provide performance data for the four codes. The effectiveness of implemented codes, in terms of bit

error rates, coding gains obtained versus the uncoded link, and code rates are the areas of study for this report.

Chapter Two is a brief overview of MFM as it pertains to a 16-20 kHz bandpass channel, the communication link under simulation. Chapter Three is a discussion of Reed-Solomon error control coding in general, including the encoding and decoding processes and other parameters necessary to define that class of codes. Also included is a description of how the codes implemented differ from conventional Reed-Solomon codes. Chapter Four describes the link simulation program and supporting programs and files, channel coding and decoding, how error control was implemented into the MFM environment, and the results obtained using the various codes compared to the uncoded channels. Chapter Five contains the conclusions and recommendations of possible ways to achieve improved performance.

II. THEORY OF MULTI-FREQUENCY MODULATION

A. BACKGROUND

Most modulation methods for bandpass channels hold the digital signal information in the amplitude and or phase of a carrier (e.g., Phase Shift Keying (PSK), Quadrature Amplitude Modulation (QAM)). Information may also be transmitted by the presence of one of two carrier frequencies (Frequency Shift Keying (FSK)). MFM is a variation on these methods. The signals are directly encoded, modulated, demodulated, and decoded using digital signal processing techniques contained in the software of the host computer.

In MFM, signals are grouped into "packets" which are arbitrarily located in the frequency spectrum, and consists of one or more bauds. An MFM packet is shown in Figure 1, on a display of frequency and time. This figure shows the nature of MFM: a structured block of time and frequency slots.

This structure makes MFM a technique that is ideally suited to allow the host computer to absorb the functions of modulation and multiplexing, demultiplexing and demodulation [Ref. 1: p. 3]. The packet consists of L bauds, each holding information on K tones. The baud length is ΔT during which K discrete tones are transmitted. In MFPSK, the phase of each tone contains the information, and in MFQAM, both the phase and magnitude of each tone contain the information. These LK subsignals form an orthogonal signal set. Each of these subsignals may be independently modulated with both phase and amplitude information.

B. GENERATION AND DEMODULATION

Generation and demodulation of MFM is based on the symmetry properties of the Fast Fourier Transform (FFT). That is, the FFT of a real-time signal corresponds to a conjugate symmetric frequency spectrum, and the inverse FFT of a conjugate symmetric frequency spectrum corresponds to a real-time signal. This means that, if only half of the frequency spectrum is used for in-phase and quadrature information, and the other half is loaded with its complex conjugate

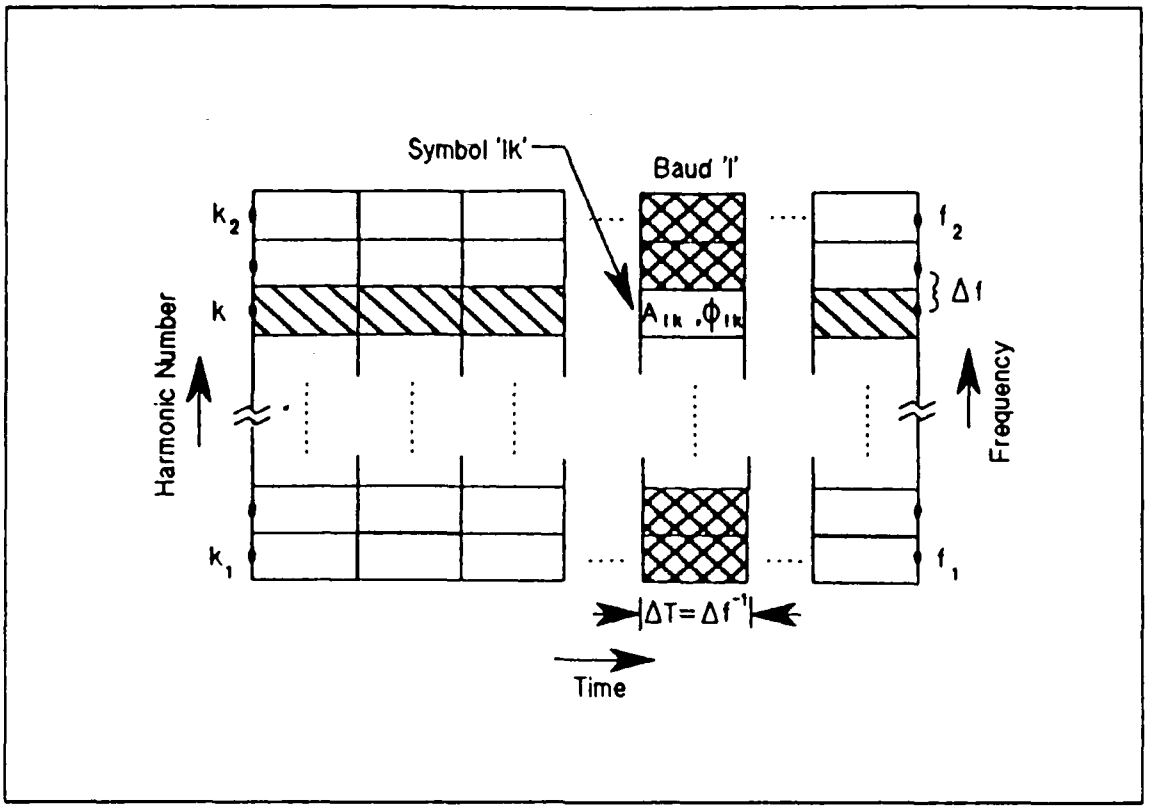

Figure 1. MFM Signal Packet: (after Ref. 1: p.3)

image, then an inverse FFT of this spectrum will result in a real-time signal. Conversely, the FFT of this real-time signal will result in the same frequency spectrum. Thus, the MFM transmitter is a device which takes an inverse FFT, and the MFM receiver is a device which takes a FFT (with appropriate D/A and A/D conversion).

In a bandpass channel, the usable portion of the frequency spectrum for information is further reduced. Current MFM applications under study at Naval Postgraduate School deal with a 16-20 kHz bandpass channel, with a D/A and A/D sampling frequency of 61440 Hz. The corresponding parameters for the MFM packets are shown in Table 1.

In the communication link under study, the baud type(s) transmitted in any given packet is known by the receiver. Because the tones containing information occur in a known band of the frequency spectrum, only those harmonic numbers

of the FFT corresponding to this band need be analyzed. This function is similar to bandpass filtering, but noise within the pass band is still present, leading to potential errors.

Table 1. MFM PACKET PARAMETERS (BANDPASS 16-20 KHZ)

BAUD TYPE	1	2	3	4	5
Space Between Tones Δf (Hz)	240	120	60	30	15
Lower Harmonic Number-k1	68	135	269	537	1073
Lower Tone (Hz)	16320	16200	16140	16110	16095
Upper Harmonic Number-k2	83	166	332	664	1328
Upper Tone (Hz)	19920	19920	19920	19920	19920
Number of Tones With Information	16	32	64	128	256
FFT Size (Baud Length)	256	512	1024	2048	4096

III. REED-SOLOMON CODES

A. INTRODUCTION

A binary block error control code with block length n which contains 2^k code words is called a *linear* (n,k) code if and only if those 2^k code words form a k-dimensional subspace of the vector space of all the n-tuples over the Galois Field GF(2). A *cyclic* code is a linear block code in which every cyclic shift of a code word (left or right) is also a code word. Cyclic codes are useful because the encoding and syndrome calculation processes can be accomplished easily through shift registers with feedback connections and because their inherent algebraic structure makes possible the use of several practical methods for decoding [Ref. 2: p. 85]. These codes are normally expressed as polynomials. A large class of powerful cyclic codes are those known as BCH codes. Their importance in coding is due to the fact that they are capable of correcting all random patterns of t errors with a simple decoding algorithm that is easily implemented [Ref. 3: p. 167].

Reed-Solomon codes form a special subclass of non-binary BCH codes. A conventional (n,k) t-error correcting Reed-Solomon code which is comprised of elements of the Galois Field GF(q) has the following structure:

- code word block length $n = q - 1$
- number of parity symbols $n - k = 2t$
- minimum distance $d_{min} = 2t + 1$

For this design, q is taken to be a power of two (i.e., $q = 2^m$). Appendix A displays a binary representation of the elements of $GF(2^4)$ and $GF(2^5)$.

B. ENCODING

If α is a primitive element of $GF(2^m)$, then the generator polynomial of this (n,k) Reed-Solomon code is given by

$$G(x) = (x + \alpha)(x + \alpha^2) \ldots (x + \alpha^{2t}) \\ = g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t-1} x^{2t-1} + x^{2t} \tag{1}$$

The generator polynomial roots can also be shifted to save circuit components [Ref. 4: p. 174]. In this case, $G(x)$ takes the form

$$G(x) = (x + \alpha^{j_0})(x + \alpha^{j_0+1}) \ldots (x + \alpha^{j_0+2t-1}), \tag{2}$$

where j_0 is any integer. The code generated by $G(x)$ consists of all polynomials with coefficients from GF(2^m), of degree $n - 1$ or less, and which are multiples of $G(x)$. The message to be encoded is a polynomial, $U(x)$, of degree $k - 1 = n - 2t - 1$ or less. In non-systematic form, the corresponding codeword polynomial is

$$V(x) = G(x)U(x) = v_0 + v_1 x + v_2 x^2 + \cdots + v_{n-1} x^{n-1} \tag{3}$$

This is clearly a multiple of $G(x)$.

Encoding in systematic form requires more computation. To structure the codewords with the information symbols preceeded by the parity symbols, $U(x)$ must first be pre-multiplied by x^{2t}. Then $x^{2t}U(x)$ is divided by $G(x)$, which yields a quotient $Q(x)$ and a remainder $B(x)$ of degree $2t - 1$ or less:

$$x^{2t}U(x) = Q(x)G(x) + B(x) \tag{4}$$

If $B(x)$ is added to $x^{2t}U(x)$, then the result is a multiple of $G(x)$ which is the systematic codeword corresponding to $U(x)$:

$$x^{2t}U(x) + B(x) = Q(x)G(x) \tag{5}$$

This polynomial division can be implemented in a shift register feedback circuit as shown in Figure 2.

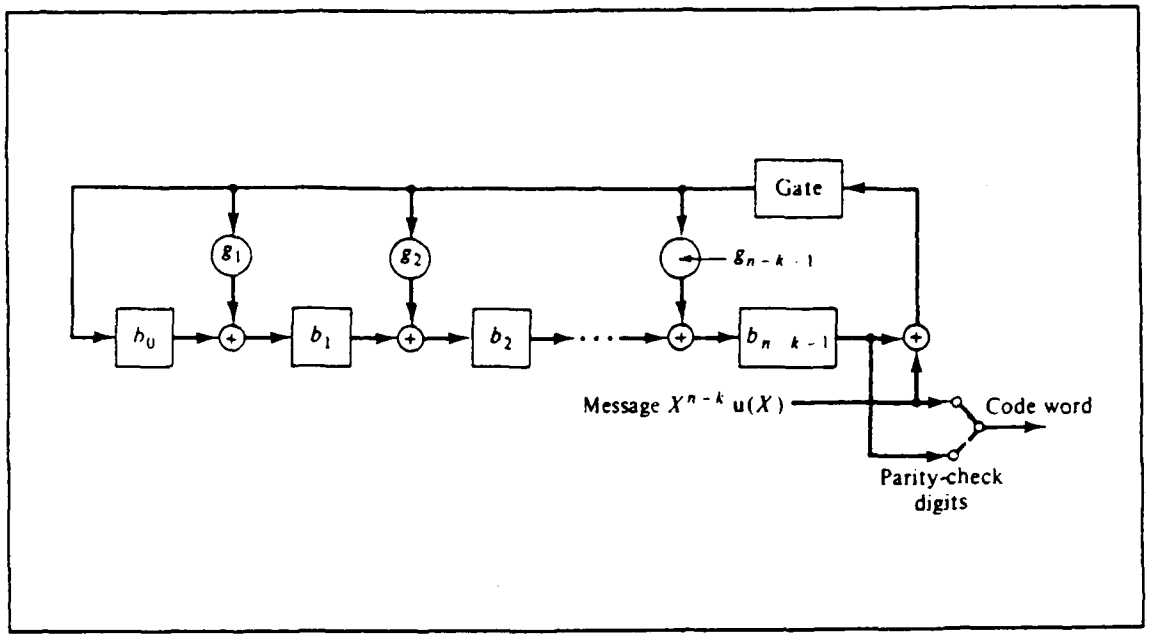

Figure 2. Systematic Encoding Circuit for an (n, k) Cyclic Code: [Ref. 2: p. 95]

Using the circuit in Figure 2, an (n, k) linear cyclic code with generator polynomial $G(x)$ as defined in equation (1) or equation (2) is encoded as follows:

1. With the gate turned on, the k information symbols $u_0, u_1, \ldots, u_{k-1}$ are shifted into the communication channel and at the same time shifted into the the encoding circuit. This automatically has the effect of pre-multiplying $U(x)$ by x^{2t}.
2. After k clock cycles, the entire message has been shifted into the channel and the $n-k$ shift registers contain the elements of the remainder polynomial.
3. The gate is opened to break the feedback connections and the switch is shifted to transfer parity symbols.

4. The parity symbols are clocked out onto the channel. These $n-k$ parity symbols together with the k information symbols form the n-length codeword.

Reed-Solomon codes in systematic form use the same encoding circuit.

C. DECODING

Decoding Reed-Solomon codes can be performed using a table-lookup method or using iterative algorithms (Berlekamp, Forney, Euclid, etc.) [Refs. 4, 5]. For binary BCH codes, these algorithms determine the location of errors (using the error location polynomial). For non-binary codes, as in the case of Reed-Solomon codes, once the error location polynomial is found, the magnitude of the error(s) must also be found in order to correct the received signal block.

Because every codeword $V(x)$ is divisible by the generator polynomial $G(x)$ and the roots of $G(x)$ are α, α^2, ..., α^{2t}, these are also the roots of $V(x)$. As such, the following equation holds for $1 \leq i \leq 2t$:

$$V(\alpha^i) = v_0 + v_1\alpha^i + v_2\alpha^{2i} + \cdots + v_{n-1}\alpha^{(n-1)i} = 0 \qquad (6)$$

or in vector form,

$$(v_0, v_1, v_2, \ldots, v_{n-1}) \bullet \begin{bmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ . \\ . \\ . \\ \alpha^{(n-1)i} \end{bmatrix} = 0 \qquad (7)$$

By forming an n by $n-k=2t$ matrix whose columns are the vectors of powers of α^i, $i=1,2,\ldots,2t$ in the previous equation, any codeword vector will be orthogonal to that matrix. The matrix thus formed is the transpose of the parity check matrix for a conventional (n,k) Reed-Solomon code. In other words, the parity check matrix for an (n,k) Reed-Solomon code is given by:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & (\alpha^2) & (\alpha^2)^2 & \ldots & (\alpha^2)^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & \ldots & (\alpha^3)^{n-1} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & (\alpha^{2t}) & (\alpha^{2t})^2 & \ldots & (\alpha^{2t})^{n-1} \end{bmatrix} \qquad (8)$$

The syndrome of a received polynomial is found from multiplying the received polynomial (in vector form) by the transpose of the parity check matrix:

$$\mathbf{s} = \mathbf{r} \bullet \mathbf{H}^{\mathrm{T}} = (\mathbf{v} + \mathbf{e}) \bullet \mathbf{H}^{\mathrm{T}} = \mathbf{e} \bullet \mathbf{H}^{\mathrm{T}}, \tag{9}$$

where $\mathbf{s}$ is the syndrome vector, $\mathbf{r}$ is the received vector, $\mathbf{v}$ is the transmitted codeword vector, $\mathbf{e}$ is the error vector, and $\mathbf{H}$ is the parity check matrix. (Recall that any codeword vector $\mathbf{v}$ is orthogonal to $\mathbf{H}^{\mathrm{T}}$.) Syndrome component s_i can also be calculated by evaluating the received polynomial, $R(x)$, at α^i, for $i = 1,2,\ldots,2t$. Once the syndrome components are found, they are used in the iterative calculations needed to find the error location polynomial, $\sigma(x)$, where

$$\begin{aligned} \sigma(x) &= (1 + \beta_1 x)(1 + \beta_2 x) \cdots (1 + \beta_v x) \\ &= \sigma_0 + \sigma_1 x + + \sigma_2 x^2 + \cdots + \sigma_v x^v \end{aligned} \tag{10}$$

Here, the roots of $\sigma(x)$ ($\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_v^{-1}$) are the error location numbers, and v is the total number of errors in the received codeword (for $v < t$). Then defining

$$\begin{aligned} Z(x) = 1 + (s_1 + \sigma_1)x + (s_2 + \sigma_1 s_1 + \sigma_2)x + \cdots \\ + (s_v + \sigma_1 s_{v-1} + \sigma_2 s_{v-2} + \cdots + \sigma_v)x^v, \end{aligned} \tag{11}$$

the error magnitude at location $\beta_l = \alpha^{j_l}$ is [Ref. 2: p. 175]:

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\prod_{i=1, i \neq l}^{\mu} (1 + \beta_i \beta_l^{-1})}. \tag{12}$$

This demonstrates the ease of implementation of the iterative algorithms; the disadvantage is that the iterative solution does not provide high-speed decoding.

On the other hand, a table-lookup method can provide high-speed decoding, but requires too much memory or overly complicated logic circuitry even for moderate code lengths [Ref. 5: p. 1]. However, some modifications to Reed-Solomon codes can allow a faster decoding procedure.

D. MODIFIED REED-SOLOMON CODES

There are several ways to modify Reed-Solomon codes to fit specific applications. The block length can be extended or shortened, and the number of parity or information symbols can be varied. The $(2^m-1,\ 2^m-2t-2)$ codes utilized in the simulation carry fewer information symbols (more parity) than the conventional codes, reducing the code rate. The benefits lie in the enhanced error correction detection capability and speed of decoding.

1. The $(2^m-1, 2^m-4)$ Reed-Solomon Codes

By replacing one of the information symbols in the conventional $(2^m-1,\ 2^m-3)$ code with a parity symbol, the single-error correcting code can also detect all double errors. This modified SEC DED code has the following characteristics:

- codeword block length $n = q - 1$
- number of parity symbols $n - k = 3$
- minimum distance $d_{\min} = 4$

Generation of this code is similar to conventional codes, except that now the generator polynomial has an additional factor:

$$G(x) = (x+1)(x+\alpha)(x+\alpha^2)$$
$$= \begin{cases} \alpha^3 + \alpha^{11}x + \alpha^{10}x^2 + x^3, & \text{for the (15,12) code} \\ \alpha^3 + \alpha^{12}x + \alpha^{11}x^2 + x^3, & \text{for the (31,28) code} \end{cases} \qquad (13)$$

The encoding circuit for these codes is still that of Figure 2. The major difference between these modified codes and the conventional codes lies in the decoding.

Reference 6 details a fast decoding technique for an extended SEC/DED Reed-Solomon code based solely on the syndrome components (i.e., iterative algorithms are not required). The same decoding technique applies to the modified codes used in the simulation, since the modified codes are shortened versions of the extended codes. For the simulation, the blocklength of the extended codes is shortened by three symbols to better match the number of tones per baud type, and to facilitate encoding.

The extended codes of Reference 6 are formed by the addition of a three by three identity matrix to the parity check matrix of the code generated by the generator polynomial of equation (13). This parity check matrix is then shortened by deleting the three right-most columns for use in the $(2^m - 1,\ 2^m - 4)$ simulation codes:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & \ldots & 1 \\ 0 & 1 & 0 & 1 & \alpha & (\alpha)^2 & \ldots & (\alpha)^{n-4} \\ 0 & 0 & 1 & 1 & \alpha^2 & (\alpha^2)^2 & \ldots & (\alpha^2)^{n-4} \end{bmatrix} \tag{14}$$

The decoding algorithm for these codes is as follows:

1. Calculate the syndrome components (in vector form, $\mathbf{s} = (s_0, s_1, s_2)$). If all are zero, decide no errors occurred.
2. If a single error occurs in one of the first three positions of the received vector (which corresponds to the identity matrix positions of **H**), then the syndrome has only one non-zero element. The error has magnitude equal to the non-zero syndrome component and location corresponding to the syndrome component number.
3. A single error in any other position satisfies

$$\frac{s_1}{s_0} = \frac{s_2}{s_1} = \alpha^{i-3}, \tag{15}$$

with the error in position i of magnitude s_0.

4. If a single syndrome component is zero, or if

$$\frac{s_1}{s_0} \neq \frac{s_2}{s_1}, \tag{16}$$

then decide that at least two symbol errors occurred.

5. Once errors are corrected, recalculate the syndrome to ensure all components are zero. If not, decide that two or more symbol errors occurred which yielded the same syndrome as a single error. (Note: this step was not included in Ref. 6 but was found necessary to detect all double errors.)

As an example, suppose that for the (15,12) code a single error of magnitude α^9 occurred in the fifth position of the received block **r**. Then the error vector is given by $\mathbf{e} = (0,0,0,0,\alpha^9,0,\ldots,0)$, and the syndrome is $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = \mathbf{e} \cdot \mathbf{H}^T = (\alpha^9, \alpha^{11}, \alpha^{13})$. Then $s_1 s_0 = s_2 s_1 = \alpha^2 = \alpha^{5-3}$, corresponding to an error of magnitude $s_0 = \alpha^9$ in the fifth position.

2. The $(2^m - 1, 2^m - 6)$ Reed-Solomon Codes

The $(2^m - 1, 2^m - 6)$ codes are obtained by making the same alteration to a conventional double-error correcting code as was the case for the SEC/DED code: sacrifice one information symbol to parity. This now allows the detection of all triple errors, and as will be shown later, the correction of some triple burst errors. The characteristics of the DEC/TED codes are:

- code word block length $n = q - 1$
- number of parity symbols $n - k = 5$
- minimum distance $d_{min} = 6$

The generator polynomial for this code also has an additional factor over the conventional code, and using shifted roots is given by:

$$G(x) = (x + \alpha^{-2})(x + \alpha^{-1})(x + 1)(x + \alpha)(x + \alpha^2)$$
$$= \begin{cases} 1 + \alpha^4 x + \alpha^{11} x^2 + \alpha^{11} x^3 + \alpha^4 x^4 + x^5, & \text{for the (15,10) code} \\ 1 + \alpha^{13} x + \alpha^{17} x^2 + \alpha^{17} x^3 + \alpha^{13} x^4 + x^5, & \text{for the (31,26) code} \end{cases} \quad (17)$$

Once again, these codes are encoded in systematic form using the circuit of Figure 2. Decoding is, however, somewhat more involved than for the SEC/DED codes.

Reference 5 gives a detailed description and proof of a fast-decoding technique for these codes which is again solely based on the syndrome of the received signal. The parity check matrix of these codes has an additional row over the conventional codes and is given by:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha^{-2} & (\alpha^{-2})^2 & \dots & (\alpha^{-2})^{n-1} \\ 1 & \alpha^{-1} & (\alpha^{-1})^2 & \dots & (\alpha^{-1})^{n-1} \\ 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & (\alpha)^2 & \dots & (\alpha)^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \dots & (\alpha^2)^{n-1} \end{bmatrix} \quad (18)$$

The steps of the decoding algorithm are as follows:

1. Compute the syndrome (in vector form, $\mathbf{s} = (s_{-2}, s_{-1}, s_0, s_1, s_2)$). If all components are zero, decide that no errors occurred.
2. If more than two components of the syndrome are zero, decide that at least three symbols are in error.
3. Compute γ_1, γ_2, and γ_3, where

$$\gamma_1 = s_1 s_{-2} + s_{-1} s_0 \quad (19)$$

$$\gamma_2 = s_2 s_{-2} + s_0^2 \quad (20)$$

$$\gamma_3 = s_0 s_1 + s_2 s_{-1}. \quad (21)$$

If $\gamma_1 = \gamma_2 = \gamma_3 = 0$ then calculate $\alpha^i = \frac{s_1}{s_0}$. Correct a single error at location i of magnitude s_0.

4. If γ_1, γ_2, and γ_3 are not all zero but at least one of them equals zero, decide that at least three symbols are in error.

5. If none of the γ_i's are zero, compute $k = c/b^2$ and the trace of k, where $b = \gamma_2/\gamma_1$, $c = \gamma_3/\gamma_1$, and the trace of k, an element of GF(2^m), is defined as

$$\mathrm{Tr}(k) = \sum_{i=0}^{m-1} k^{2^i}. \tag{22}$$

If $\mathrm{Tr}(k) = 1$, decide that at least three symbol errors occurred.

6. If $\mathrm{Tr}(k) = 0$, assume that a double symbol error occurred and solve $y^2 + by + c = 0$ to find the roots α^i and α^j. Then the first error is in the i^{th} position with magnitude

$$e_1 = \frac{(s_0\alpha^j + s_1)}{b}, \tag{23}$$

and the second error is in the j^{th} position with magnitude

$$e_2 = s_0 + e_1. \tag{24}$$

7. Once errors are corrected, recalculate the syndrome to ensure all components are zero. If not, decide that at least three symbol errors occurred which yielded the same syndrome as a double error. (Note: this step was not included in Ref. 5 but was found necessary to detect all triple errors.)

Reference 5 does not examine the code's capability of triple error correction, but the correction of certain triple burst-errors is possible. Suppose that three errors of magnitude e_1 occur in consecutive positions of the received signal, the first at position i, $i = 0,1,\ldots,n-1$. (Note that wrap-around errors can also be corrected, e.g., errors in the $(n-1)^{st}$, 0^{th} and 1^{st} positions.) The error polynomial $E(x)$ is of the form

$$E(x) = e_1x^i + e_1x^{i+1} + e_1x^{i+2}, \tag{25}$$

or in vector form, $\mathbf{e} = (0,0,...,0,e_1,e_1,e_1,0,...,0)$. Multiplying the error vector by the transpose of the parity check matrix gives a syndrome vector of the form

$$\mathbf{s} = \begin{bmatrix} s_{-2} \\ s_{-1} \\ s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} e_1(\alpha^{-2i} + \alpha^{-2(i+1)} + \alpha^{-2(i+2)}) \\ e_1(\alpha^{-i} + \alpha^{-(i+1)} + \alpha^{-(i+2)}) \\ e_1 \\ e_1(\alpha^{i} + \alpha^{i+1} + \alpha^{i+2}) \\ e_1(\alpha^{2i} + \alpha^{2(i+1)} + \alpha^{2(i+2)}) \end{bmatrix} \tag{26}$$

From these relations, it is clear that if this type of triple error occurs, then the error locations and magnitude can be calculated solely from the syndrome if the following simplified conditions exist:

$$\begin{bmatrix} s_{-2} \\ s_{-1} \\ s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} e_1\alpha^{-2i}(1 + \alpha^{-2} + \alpha^{-4}) \\ e_1\alpha^{-i}(1 + \alpha^{-1} + \alpha^{-2}) \\ e_1 \\ e_1\alpha^{i}(1 + \alpha + \alpha^{2}) \\ e_1\alpha^{2i}(1 + \alpha^{2} + \alpha^{4}) \end{bmatrix} \tag{27}$$

Note that the addition of elements of GF(2^m) vary with the value of m, so the exact relations between syndrome components and error locations vary with the code used. As an example, for the (15,10) code with elements from GF(2^4), the relations will be

$$\mathbf{s} = \begin{bmatrix} s_{-2} \\ s_{-1} \\ s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} e_1\alpha^{-2i}(\alpha) \\ e_1\alpha^{-i}(\alpha^8) \\ e_1 \\ e_1\alpha^{i}(\alpha^{10}) \\ e_1\alpha^{2i}(\alpha^5) \end{bmatrix} = \begin{bmatrix} e_1\alpha^{1-2i} \\ e_1\alpha^{8-i} \\ e_1 \\ e_1\alpha^{10+i} \\ e_1\alpha^{5+2i} \end{bmatrix} \tag{28}$$

Then to check for this type of error in the (15,10) code, the syndrome components must satisfy $s_{-1}s_1 = (s_0)^2\alpha^3$ and $s_{-2}s_2 = (s_0)^2\alpha^6$. The error magnitude is then the value of s_0 and the starting location can be obtained from any of the other syndrome components, say by dividing s_1 by $s_0\alpha^{10}$ and taking the exponent of α. This capability allows correction of an additional 240 errors [15 triplets x 16 elements in GF(2^4)] for the (15,10) code and 992 errors [31 triplets x 32 elements in GF(2^5)] for the (31,26) code.

IV. SIMULATION AND RESULTS

A. BACKGROUND

The MFM link is simulated in the Fortran-77 program "MFMLINK". For a given baudtype and code (uncoded 16-QAM, uncoded 32-QAM, and (15,10), (15,12), (31,26) and (31,28) Reed-Solomon codes), the program chooses random data from GF(2^4) for 16-QAM or from GF(2^5) for 32-QAM. Uncoded QPSK was also included for comparison with the QAM results.

For the cases with error control, this data is encoded into Reed-Solomon code word blocks. For each baud, the elements of GF(2^m) are then transformed into the appropriate QAM symbol with each tone representing one element of GF(2^m), loaded into the frequency spectrum (with conjugate symmetry) and transmitted with the inverse-FFT. At this point (in the time domain), white Gaussian noise is added. The FFT is then taken, and each FFT sample within the information band (from tones k_1 to k_2) is transformed into an element of GF(2^m).

For the coded cases, these $k_2 - k_1 + 1$ symbols are divided into code word blocks, and each block is decoded separately. The output of the link is then compared to the input for error calculation. A block diagram of the link is shown in Figure 3. A summary of the Reed-Solomon codes used is shown in Table 2, with code rate $R = k/n$ and spectral efficiency $\eta = R\eta_{unc}$ (η_{unc} is the uncoded MF-QAM spectral efficiency).

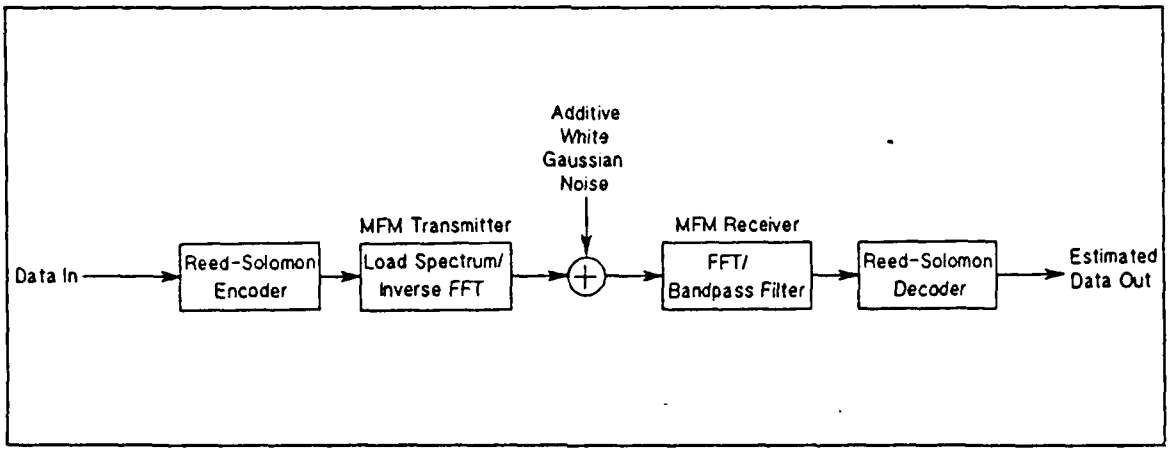

Figure 3. Simulation MFM Link

Table 2. SUMMARY OF CODES IN SIMULATION

Code (n,k)	(15,12)	(15,10)	(31,28)	(31,26)
Block Length (n)	15	15	31	31
Information Symbols (k)	12	10	28	26
Parity Symbols $(n-k)$	3	5	3	5
d_{min}	4	6	4	6
R	0.75	0.67	0.90	0.84
η(Bits/Hz)	3.0	2.67	4.52	4.19
Error Correction/Detection	SEC/DED	DEC/TED*	SEC/DED	DEC/TED*
* can correct some triple burst errors				

B. CHANNEL CODING AND DECODING

The 16-QAM constellation consists of two magnitude rings and eight phases. The magnitude rings are of amplitudes $1X$ and $2X$, with the value of X variable. Data from simulation runs was collected with $X = 100$. The phases employed were offset $\pm 22.5^{\circ}$ from the axes. Figure 4 is a diagram of the 16-QAM

constellation and Appendix A demonstrates the one-to-one relation between Galois field elements and QAM constellation elements for both 16-QAM and 32-QAM. Decoding for this constellation is accomplished for each sample in the information band of the FFT as follows:

1. Determine the quadrant of the received complex signal by the sign of its real and imaginary parts.
2. Determine which of the two phases in the quadrant is closest to the signal. This is done by comparing the absolute value of the real part to that of the imaginary part. If that of the real part is greater, choose the phase closest to the real axis. If not, choose the phase closest to the imaginary axis.
3. Finally, estimate the correct magnitude: compare the received magnitude with $1.5X$. If the received magnitude is greater than $1.5X$, assume that the received signal is actually the constellation element on the outer ring. If not, assume it is the element on the inner ring.

Two 32-QAM constellations were used in the simulation. The first is Gray coded and consists of four magnitude rings and eight phases. The magnitude rings are of amplitudes $1X$, $2X$, $3X$ and $4X$, once again with X variable. Data for 32-QAM was collected with $X = 20$. The eight phases are the same as for 16-QAM, $\pm 22.5^{\circ}$ offset from the axes. Figure 5 is a diagram of this Gray-coded constellation. The second 32-QAM constellation is identical to the first except that the rings of magnitude $2X$ and $4X$ are shifted clockwise 22.5° to increase the minimum distance between constellation elements. Figure 6 is a diagram of the modified 32-QAM constellation. Channel decoding for both 32-QAM constellations uses a different algorithm than 16-QAM, one which finds that element of the constellation that is closest in distance to the received signal (a form of maximum-likelihood decoding). It is described as follows, with r_x being a received complex sample in the information band of the FFT and q_i, $i = 0,1,...,31$ being the elements of the 32-QAM constellation:

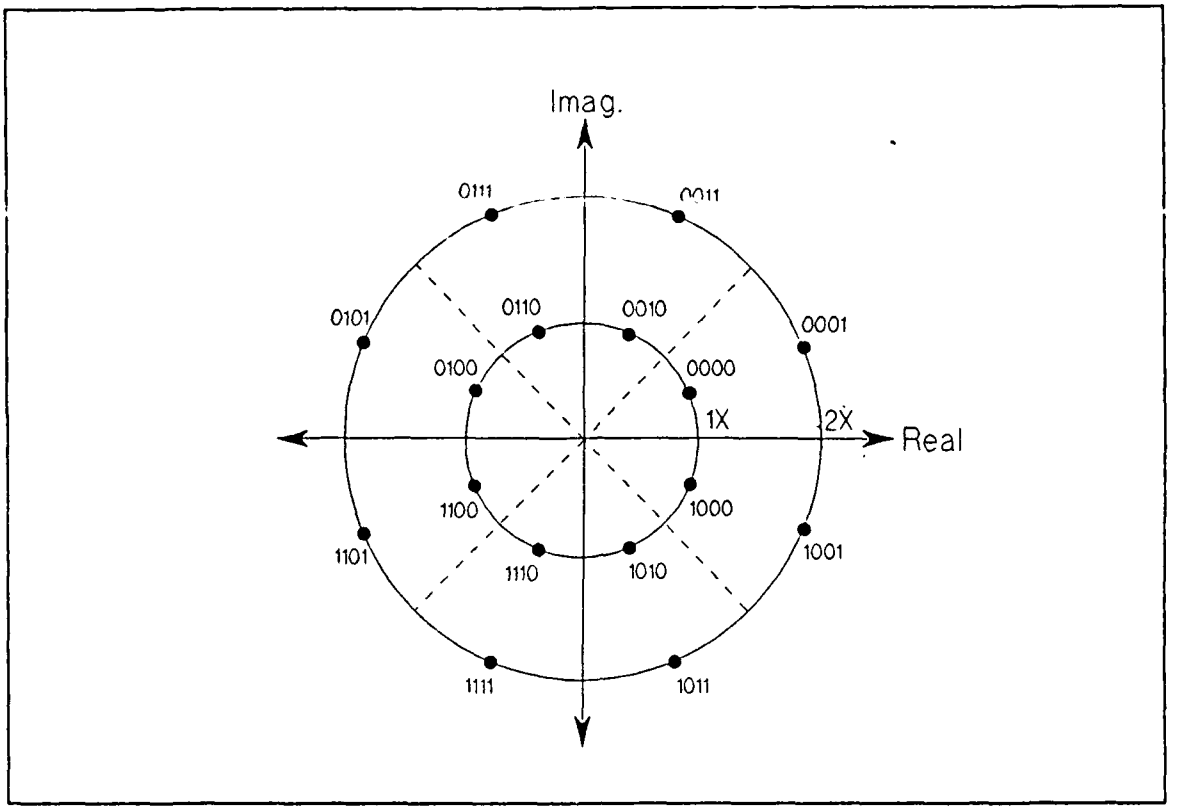

Figure 4. 16-QAM Constellation

1. Let $y_{min} = \sqrt{(\mathrm{Re}(r_x) - \mathrm{Re}(q_0))^2 + (\mathrm{Im}(r_x) - \mathrm{Im}(q_0))^2}$, the length of a line segment drawn between r_x and q_0 on the real-imaginary axis.
2. Compare y_{min} to $y_i = \sqrt{(\mathrm{Re}(r_x) - \mathrm{Re}(q_i))^2 + (\mathrm{Im}(r_x) - \mathrm{Im}(q_i))^2}$ for each successive element of the constellation. If y_i is less than y_{min}, replace the value of y_{min} with the value of y_i, store the identity of the constellation element which gave the smaller value (denoted by the subscript i), and try the next element.
3. After comparison to all elements of the constellation, i identifies the closest element. Assume that the element i is the actual symbol.

The QPSK constellation consists of four phases: one each at $\pm 45^{\circ}$ and $\pm 135^{\circ}$. Data was collected with these points having magnitude ten. Channel decoding consists of determining the quadrant of the received samples based on the

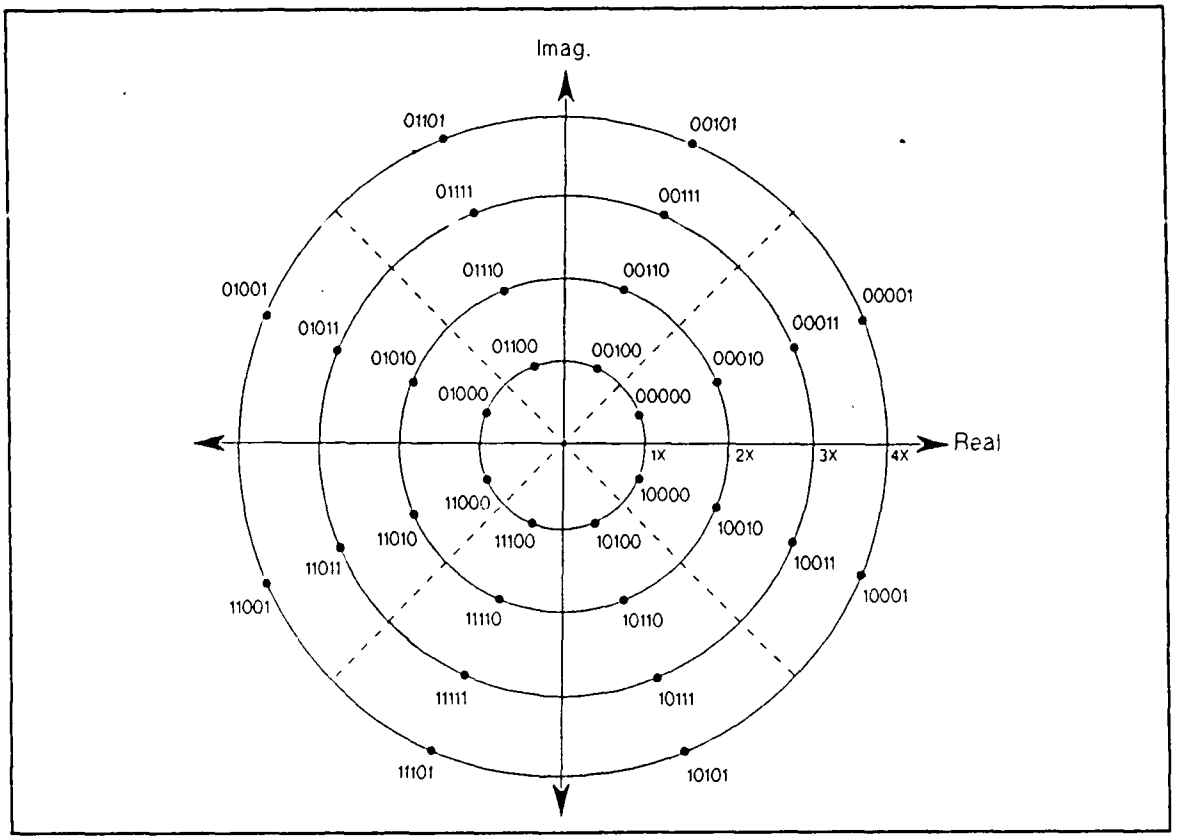

Figure 5. Gray Coded 32-QAM Constellation

signs of their real and imaginary parts. The received phase is assumed to be that which lies within the received quadrant.

C. SUPPORTING FILES AND PROGRAMS

Prior to writing an MFM link program, two data files were created ("GF16 DATA" and "GF32 DATA") which contain the binary representations of the elements of GF(2^4) and GF(2^5), respectively. These files form the basis of the Reed-Solomon computations in the link. In addition, three data files ("QAM16", "QAM32" and "QAM32A") were created which outlined the QAM constellations used, containing magnitude and phase information on each element.

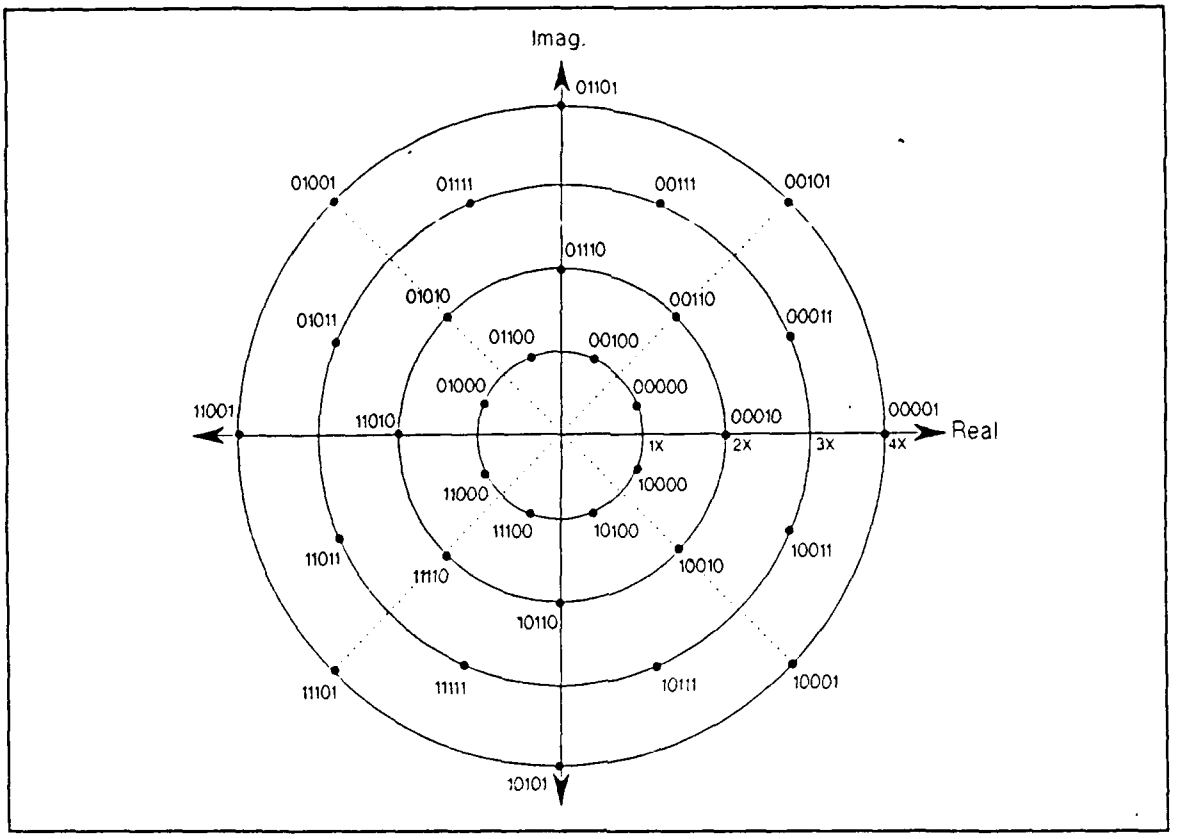

Figure 6. Modified 32-QAM Constellation

Several other programs were written to support the operation of MFMLINK. The program "GALOIS" reads the m-bit representations of a Galois field and outputs the GF(2^m) addition table. GALOIS adds each element bit-by-bit (modulo-2) and then compares the result against each element of the input data file to determine which element of GF(2^m) it corresponds to. This addition table is output to the data file "GFA16" or "GFA32" as appropriate. The program "BITDIFF" reads in the m-bit representations of GF(2^m) and compares each element bit-by-bit to determine the number of places in which each differs for use in calculating the number of bit errors. The output of BITDIFF is a GF(2^m)

"difference" table, either "GFD16" or "GFD32". The 16-element difference table for the QPSK symbols was entered separately in the file "QPSKDI". "TRACE" reads in the Galois field addition tables created by "GALOIS" and computes the trace of each element (see eqn. 20). Finally, "QAMCOM" reads in the magnitude and phase information of each QAM constellation and calculates the corresponding real and imaginary components. This output is placed in data file "QAMRI16", "QAMRI32 or "QAMRI32A", for use in channel encoding and decoding. The real and imaginary components for the QPSK constellation were entered separately in the data file "QPSKRI".

The outputs of MFMLINK are written into the data files "RSRSLTS" and another file with a variable name, depending on the coding scheme used. RSRSLTS contains information on code chosen, total symbol and bit errors, number of bauds run, and (if desired) the actual random data symbols input, transmitted code words, received signals, and corrected code words. The file with the variable name contains only the SNRs used in the run and resulting bit error rates for use in plotting, and is named according to the code and baud type of the run (e.g., "C0BT1" corresponds to code "0" (uncoded 16-QAM) and baud type "1").

D. MFMLINK

MFMLINK is subroutine based, for ease in altering parameters or algorithms. One of the difficulties in adapting a finite Galois field into a software simulation is the use of non-binary field arithmetic, the addition and multiplication of the powers of primitive element α. In the program (and supporting files), this was handled by identifying field elements in terms of its power of α plus one,

except for the zero element which is always "0" (i.e., $0 \rightarrow 0$, $1 = \alpha^0 \rightarrow 1$, $\alpha^1 \rightarrow 2$, etc.). With this method of element identification for GF(2^m), field multiplication/division of elements other than zero is carried out by addition/subtraction modulo $n = 2^m - 1$.

A program run starts with the user choosing the type of coding scheme and baud, and, if error control is utilized, whether code words with more errors detected than the code can correct should be erased. SNRs and the number of bauds for each SNR are fixed in the program and must be edited in order to change. Once the user inputs these parameters, subroutine "BAUDS" reads the FFT size and its logarithm base two, k_1, k_2, and the number of code words per baud (when error control is used). Subroutine "DATA" then reads in the Galois addition and difference tables, element traces, and QAM constellation characteristics. Once this background data is read, the run commences.

If an uncoded case is chosen, subroutine "NOCODE" provides random Galois field elements as data and the corresponding QAM or QPSK constellation elements, and loads the inverse-FFT array for transmission. For coded cases, the main program determines the random Galois field data for a code word, which is then encoded in Reed-Solomon systematic form (with subroutine "ENCODE"). This process is repeated until the baud is filled with Reed-Solomon code words, with each tone representing one element of the Galois field in use. The main program then loads the corresponding inverse-FFT array for transmission.

Subroutine "BITREV" places the array in bit-reversed order for use in the subroutine "INVFFT" which takes the inverse-FFT of the transmission array

using the decimation-in-time algorithm. Both BITREV and INVFFT were copied from the digital signal processing files of Professor Robert D. Strum at NPS.

Once a baud has been transmitted, noise is added to each sample of the time domain signal. First, signal power is calculated from the sum of the squares of the time domain signal samples (a real-valued sequence). Then noise power (or variance, since the noise is fixed to have zero mean) is calculated from

$$\sigma_w^2 = \frac{1}{(N_s - 1)} \frac{\sum_{i=0}^{N_s-1} T_x(i)^2}{10^{(SNR-\eta)/10.0}} \tag{29}$$

where σ_w^2 is the noise power, N_s is the FFT size, T_x is the inverse-FFT array (transmitted signal), and η is given by

$$\eta = 10 \log_{10}\left[\frac{\frac{N_{FFT}}{2}}{k_2 - k_1} \right] \tag{30}$$

The quantity η corrects the value of SNR to be the ratio of signal power to noise power in the frequency band from k_1 to k_2, rather than having the signal power concentrated in this frequency interval and noise power spread uniformly throughout the spectrum. Once the noise variance is calculated, subroutine "GAUSS" calculates random samples of AWGN with zero mean and variance σ_w^2 to add to each sample of the time domain signal, forming the received signal samples.

The subroutines BITREV and "FFT" (FFT is also copied from Professor Strum's files) function as the MFM receiver. Each received complex sample in the

information band from k_1 to k_2 is then channel decoded into an element of the appropriate Galois field by the subroutines "WHAT16" (for 16-QAM), "WHAT32" (for 32-QAM) or "WHATQP" (for QPSK). For uncoded cases, these decoded symbols are compared with the input symbols to determine symbol and bit errors. For runs with error control, after channel decoding each estimated code word is then decoded using the appropriate algorithm of Chapter III. First, each estimated codeword's syndrome is calculated (with subroutine "SYNDRO"). If any errors are detected, either subroutine "DECONE" (for SEC codes) or "DECTWO" (for DEC codes) are called to correct errors, if possible. If code word erasure was called for by the user, these decoding subroutines call "RUBOUT" to erase code words that cannot be corrected. After Reed-Solomon decoding, the input code words are compared to the corrected output to determine the number of symbol and bit errors. Finally, subroutine "OUTPUT" sends pertinent output information to the data file RSRSLTS.

E. RESULTS

The primary results of the simulation are displayed in the plots of bit error rate versus SNR and bit error rate versus E_B/N_0 . At lower values of SNR (or E_B/N_0), accurate results were obtained by performing runs with 10,000-20,000 bits. At higher values of SNR, as the number of errors diminished, over 100,000 bits were required. The results of the simulation runs are plotted in the eight graphs included in the following pages. The graphs are in pairs and reflect the bit error rates as a function of either SNR or E_B/N_0 . The plots with E_B/N_0 are derived from those with SNR based on the following relationship:

$$SNR_{WB} = \left(\frac{E_B}{N_0}\right) m, \tag{31}$$

where SNR_{WB} is the wide band SNR (that used as an input to the program) and m is the number of bits per tone for a particular modulation scheme. The calculation of E_B/N_0 includes the energy in all bits carried on a tone, both information and parity, since the bit error calculations include errors in both data and parity symbols. Figure 7 is a graph of the uncoded modulation schemes' performances on the link in terms of Bit Error Rate vs. SNR and also shows the same performance in terms of E_B/N_0 . Figure 8 contains similar plots showing the performance of the two Reed-Solomon codes used with 16-QAM against that of uncoded 16-QAM and uncoded QPSK. Figures 9 and 10 show the performance of the two Reed-Solomon codes used with both 32-QAM constellations against their respective uncoded cases. A summary of the BER performances of the error control codes over the corresponding uncoded cases is contained in Table 3 following the performance graphs.

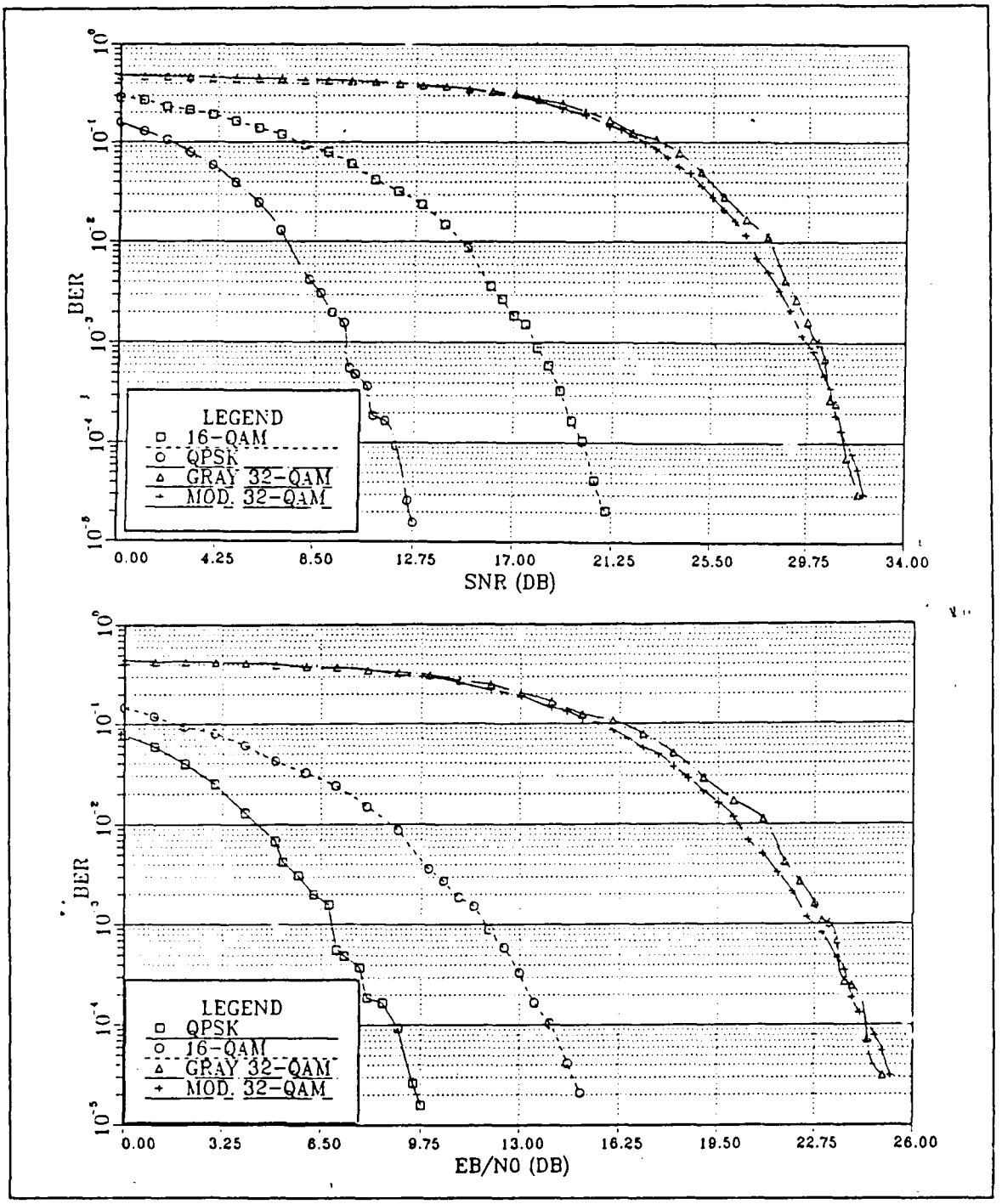

Figure 7. Uncoded Bit Error Rates: Top: BER vs. SNR. Bottom: BER vs. E_B/N_0

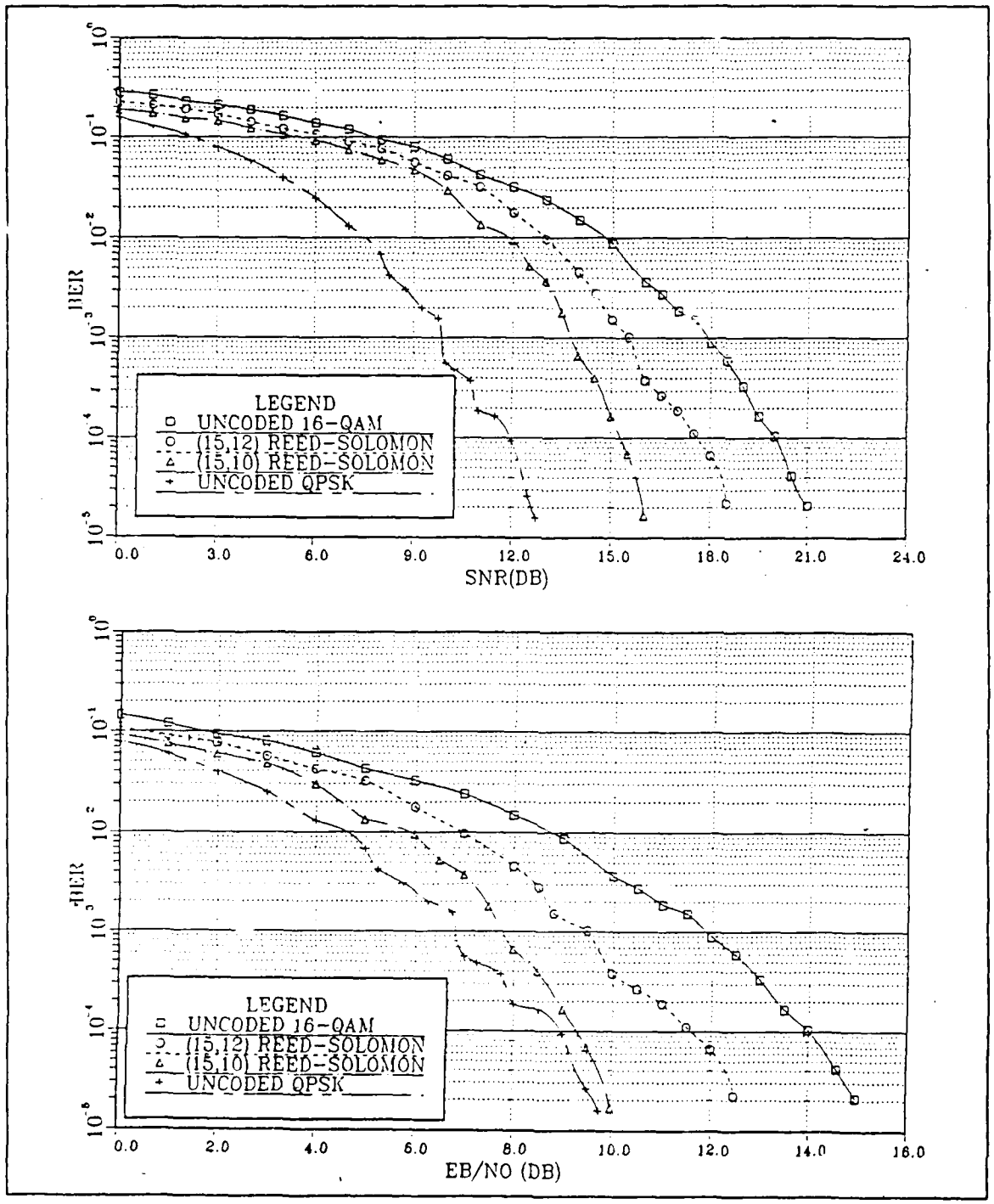

Figure 8. 16-QAM Bit Error Rates: Top: BER vs. SNR. Bottom: BER vs. E_B/N_0

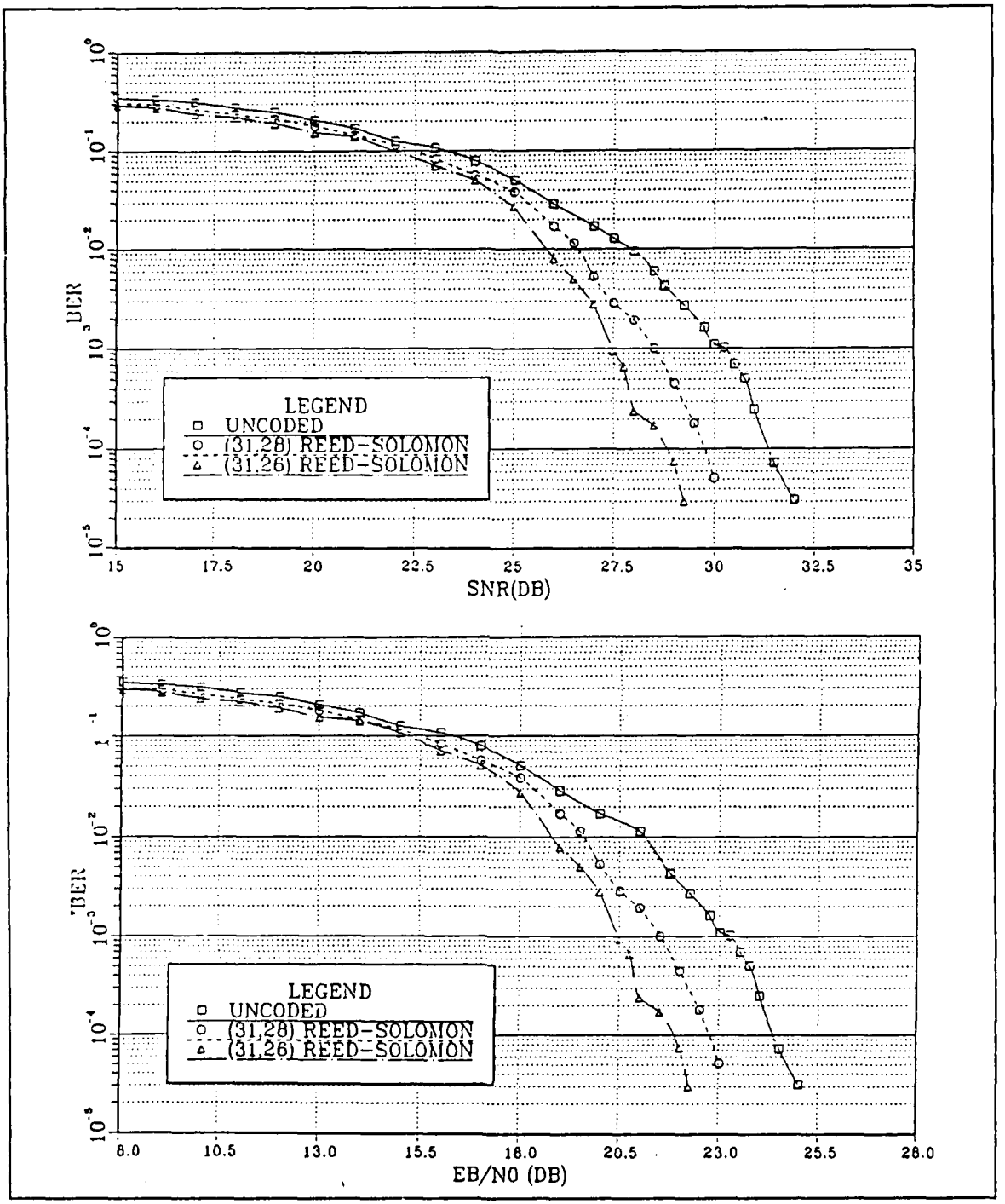

Figure 9. Gray Coded 32-QAM Bit Error Rates: Top: BER vs. SNR. Bottom: BER vs. E_B/N_0.

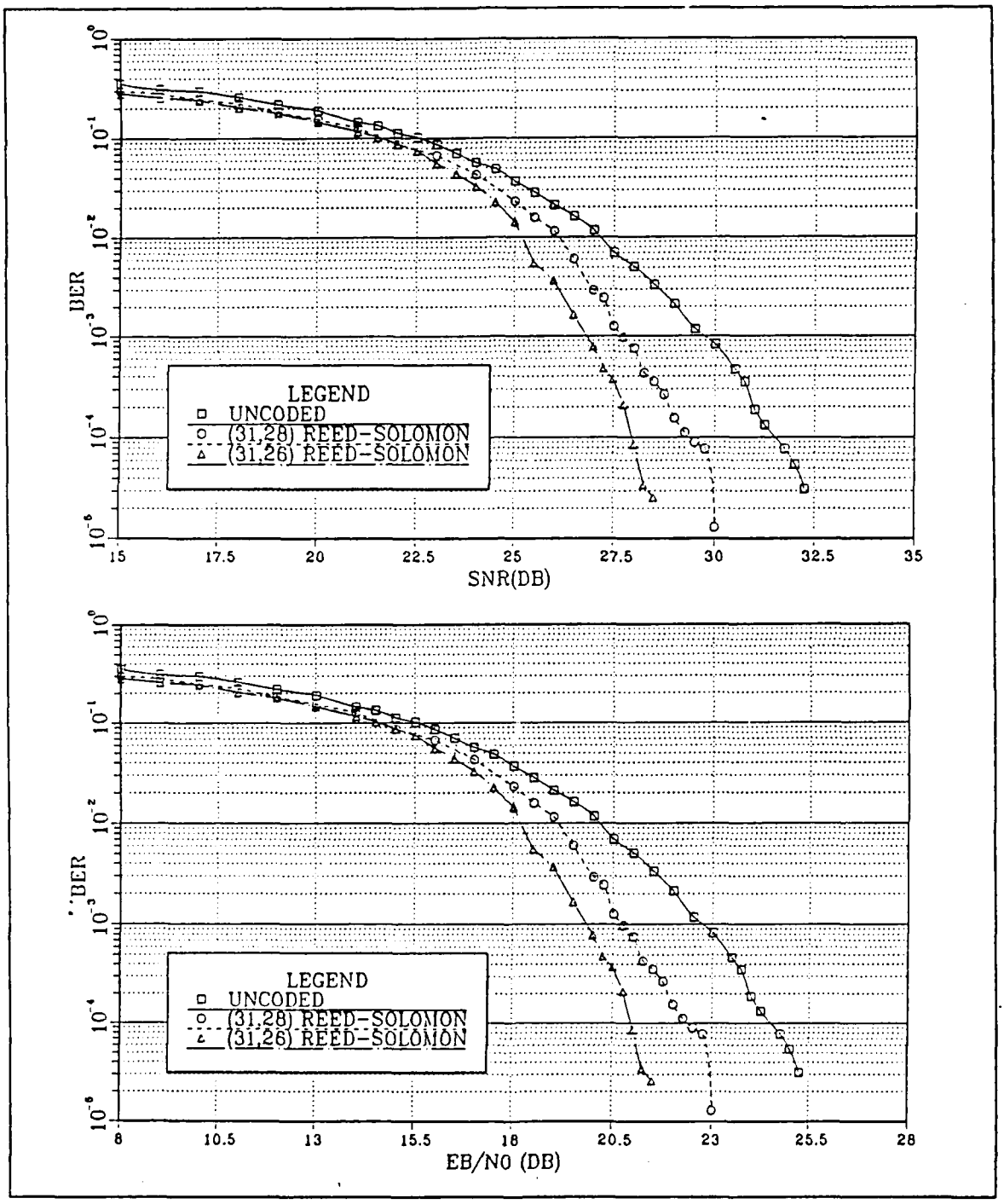

Figure 10. Modified 32-QAM Bit Error Rates: Top: BER vs. SNR. Bottom: BER vs. E_b/N_0.

Table 3. ERROR CONTROL PERFORMANCE SUMMARY

	16-QAM					
Code	(15,12)			(15,10)		
BER	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}
Coding Gain (approx.)	2.5	2.5	2.5	4.0	4.7	5.0
	Gray Coded 32-QAM					
Code	(31,28)			(31,26)		
BER	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}
Coding Gain (approx.)	1.4	1.5	1.5	2.4	2.6	2.6
	Modified 32-QAM					
Code	(31,28)			(31,26)		
BER	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}
Coding Gain (approx.)	2.1	2.1	2.5	2.9	3.5	3.8

V. CONCLUSIONS AND RECOMMENDATIONS

The simulation MFMLINK is the first implementation of error control coding in multi-frequency modulation at NPS. As such, the codes utilized may not be the optimal codes for the given link parameters, but do provide good insight into the benefits of error control coding for MFM.

First, the enhanced error detection capabilities of the modified Reed-Solomon codes in the simulation support the introduction of some type of Automatic Repeat Request (ARQ) strategy into the MFM environment. More specifically, a *type I hybrid ARQ scheme* is called for as described in Reference 2. Using this scheme with MFM, any correctable errors which are detected are corrected and if uncorrectable errors are detected, the receiver requests retransmission of particular erroneous codewords or bauds. The proper combination of ARQ and FEC can provide a higher information throughput than one with ARQ alone, and provide higher system reliability than one with FEC alone [Ref. 2: p.478].

Second, although the decoding time for any given codeword or baud was not measured, the use of the decoding algorithms presented for the modified Reed-Solomon codes (which are solely based on the calculated syndromes) are simpler to implement in software than the iterative algorithms for conventional codes and felt to be less time consuming to run. The loss in information rate due to surrendering an information symbol to parity is well worth the time saved in the decoding process.

Finally, based on the graphs presented at the end of Chapter 4, the coding gains obtained at bit error rates of 10^{-3} and 10^{-4} for each code has proven the value of the codes used. Large gains were evident using error control with 16-QAM modulation at a moderate cost in code rate. Smaller gains were evident using error control with 32-QAM modulation, particularly with the Gray-coded constellation, but at only a slight reduction in code rate. In any case, the loss in code rate due to the addition of parity symbols for error control is worth the coding gains achieved.

Since this simulation was the first venture into MFM error control implementation at NPS, several alterations to the simulation may provide improved error control performance. One might be to change the methods of channel decoding, particularly in the case of 32-QAM. The approach used with 16-QAM may have improved performance over the maximum-likelihood approach. Constellation geometries with elements farther apart may improve performance, as seen in the results obtained with the 32-QAM constellations. Another alteration might be to vary the modulation technique used with the error control codes in the simulation. For example, using QPSK and the length 15 Reed-Solomon codes with each tone carrying two bits, two consecutive tones would represent one element of $GF(2^4)$ and 30 tones would represent a complete codeword. Interleaving the codeword symbols between bauds is another possible alteration, e.g., having a 16 tone information band with each tone representing one symbol from 16 separate codewords would correspond to the transmission of 16 codewords in a 15 baud packet. Other methods of error control are also of interest, particularly the use of convolutional/trellis codes and concatenated codes.

APPENDIX A. BINARY REPRESENTATIONS OF GALOIS FIELDS

Table 4. ELEMENTS OF GF(2^4)

Primitive Polynomial: $P(x) = 1 + x + x^4$	
Element	Binary Representation
0	0000
1	1000
α	0100
α^2	0010
α^3	0001
α^4	1100
α^5	0110
α^6	0011
α^7	1101
α^8	1010
α^9	0101
α^{10}	1110
α^{11}	0111
α^{12}	1111
α^{13}	1011
α^{14}	1001

Table 5. ELEMENTS OF GF(2^5)

Primitive Polynomial: $P(x) = 1 + x^2 + x^5$	
Element	Binary Representation
0	00000

1	10000
α	01000
α^2	00100
α^3	00010
α^4	00001
α^5	10100
α^6	01010
α^7	00101
α^8	10110
α^9	01011
α^{10}	10001
α^{11}	11100
α^{12}	01110
α^{13}	00111
α^{14}	10111
α^{15}	11111
α^{16}	11011
α^{17}	11001
α^{18}	11000
α^{19}	01100
α^{20}	00110
α^{21}	00011
α^{22}	10101
α^{23}	11110
α^{24}	01111
α^{25}	10011
α^{26}	11101
α^{27}	11010
α^{28}	01101
α^{29}	10010
α^{30}	01001

APPENDIX B. SUPPORTING PROGRAMS

A. GALOIS

```
C*************************************************************************
C  THIS PROGRAM WILL READ THE M-BIT REPRESENTATIONS OF GF(2**M) FROM    *
C      A DATA FILE, THEN CALCULATE THE ADDITION TABLE ((N+1)*(N+1)) AND *
C      WRITE THE RESULTS INTO A DATA FILE GFA(N+1).                     *
C                                                                       *
C*************************************************************************

        INTEGER P(0:15),G(0:15,0:7),A(0:15,0:15,0:7),GFA(0:15,0:15)
        INTEGER S,N,M

        CALL EXCMS('FILEDEF 8 DISK GF16 DATA (PERM')
        OPEN(7,FILE='GFA16',ACCESS='SEQUENTIAL',FORM='FORMATTED',STATUS=
     +       'UNKNOWN')

C       OPEN(8,FILE='GF16',ACCESS='SEQUENTIAL',FORM='FORMATTED',STATUS
C    +       ='UNKNOWN')

        N=15
        M=4

C     READ IN THE M-BIT REPRESENTATIONS FROM DATA FILE

        DO 10 I = 0,N
              READ(8,99) P(I),(G(I,J),J=0,M-1)
              WRITE(5,99) P(I),(G(I,J),J=0,M-1)
 10     CONTINUE
 99     FORMAT(1X,I2,4I2)

C     ADD THE M-BIT ELEMENTS...

        DO 20 I=0,N
              DO 30 J=0,N
                    DO 40 K=0,M-1
                          S=G(I,K)+G(J,K)
                          A(I,J,K)=MOD(S,2)
 40                 CONTINUE
 30           CONTINUE
 20     CONTINUE

C     NOW FIGURE OUT WHICH POWER OF ALPHA THESE M BITS REPRESENT...

        DO 50 I=0,N
              DO 60 J=0,N
                    DO 70 K=0,N
                          DO 80 L=0,M-1
                            IF (A(I,J,L).NE.(G(K,L))) THEN
                                GO TO 70
                            ENDIF
```

```
 80                      CONTINUE
                         GFA(I,J)=P(K)
                         GO TO 60
 70                  CONTINUE
 60             CONTINUE
 50      CONTINUE

C     NOW WRITE THESE POWERS OF ALPHA INTO THE ADDITION TABLE...


       DO 90 I=0,N

                    WRITE(7,900) (GFA(I,J),J=0,15)
C              THE FOLLOWING LINES ARE FOR LARGER FIELD CALCULATIONS...
C                   WRITE(7,900) (GFA(I,J),J=16,31)
C                   WRITE(7,900) (GFA(I,J),J=32,47)
C                   WRITE(7,900) (GFA(I,J),J=48,63)
C                   WRITE(7,900) (GFA(I,J),J=64,79)
C                   WRITE(7,900) (GFA(I,J),J=80,95)
C                   WRITE(7,900) (GFA(I,J),J=96,111)
C                   WRITE(7,900) (GFA(I,J),J=112,127)
C                   WRITE(7,900) (GFA(I,J),J=128,143)
C                   WRITE(7,900) (GFA(I,J),J=144,159)
C                   WRITE(7,900) (GFA(I,J),J=160,175)
C                   WRITE(7,900) (GFA(I,J),J=176,191)
C                   WRITE(7,900) (GFA(I,J),J=192,207)
C                   WRITE(7,900) (GFA(I,J),J=208,223)
C                   WRITE(7,900) (GFA(I,J),J=224,239)
C                   WRITE(7,900) (GFA(I,J),J=240,255)
 90    CONTINUE
 900   FORMAT(16(1X,I2))

       ENDFILE(7)
       CLOSE(7)
       END
```

B. BITDIFF

```
C*****************************************************************************
C   THIS PROGRAM READS IN THE BIT REPRESENTATION OF A GALOIS FIELD AND *
C     CALCULATES THE # OF BITS IN WHICH EACH ELEMENT DIFFERS.  THEN IT  *
C     WRITES THE RESULTS IN A TABLE FOR USE IN FIGURING BIT ERROR RATE *
C*****************************************************************************

      INTEGER ALPHA(0:31),BIT(0:31,0:5),GFD(0:31,0:31),N,ROWS,DIFF

      CALL EXCMS('FILEDEF 9 DISK GF32 DATA (PERM')

      OPEN (7,FILE='GFD32',FORM='FORMATTED',ACCESS='SEQUENTIAL',
     +       STATUS='UNKNOWN')

      N=31
      POWER=5
      ROWS=(N+1)/16

      DO 10 I=0,N
            READ(9,*)ALPHA(I),(BIT(I,J),J=0,POWER-1)
10    CONTINUE
      DO 20 I=0,N
         DO 30 J=0,N
            DIFF=0
            DO 40 K=0,POWER-1
               IF (BIT(I,K).NE.BIT(J,K)) THEN
                  DIFF=DIFF+1
               ENDIF
40          CONTINUE
            GFD(I,J)=DIFF
30       CONTINUE
20    CONTINUE

      DO 50 I=0,N
         WRITE(7,1000)(GFD(I,J),J=0,15)
         WRITE(7,1000)(GFD(I,J),J=16,31)
50    CONTINUE
1000  FORMAT(16(I2,1X))
      END
```

C. TRACE

```
C*****************************************************************************
C   THIS IS THE TRACE FINDER PROGRAM                                          *
C*****************************************************************************
C
C
      INTEGER A,B,C,D,GFA(0:31,0:31),N

      CALL EXCMS('FILEDEF 9 DISK GFA32 FILE (PERM')
      OPEN(6,FILE='TR32',FORM='FORMATTED',ACCESS='SEQUENTIAL',
     +     STATUS='UNKNOWN')

C-------------------------------------------------------------------
C  READ IN GF(2**M) ADDITION TABLE...
      N=31

      DO 10 I=0,31
            READ(9,*) (GFA(I,J),J=0,15)
            READ(9,*) (GFA(I,J),J=16,31)
 10   CONTINUE
C-------------------------------------------------------------------
      WRITE(6,1001)
 1001 FORMAT('0',1X,'0')
      DO 55 I=2,31
            A=MOD(2*(I-1),N)+1
            A=GFA(I,A)
            B=MOD(4*(I-1),N)+1
            B=GFA(A,B)
            C=MOD(8*(I-1),N)+1
            C=GFA(B,C)
            D=MOD(16*(I-1),N)+1
            WRITE(6,1002)GFA(C,D)
 55   CONTINUE
 1002 FORMAT(1X,I2)
      END
```

D. QAMCOM

```
C*****************************************************************************
C     QAM COMPUTER- THIS PROGRAM READS IN MAGNITUDE AND PHASE INFO       *
C           ON A QAM CONSTELLATION AND CONVERTS IT TO A REAL AND         *
C           IMAGINARY PART WHICH IS STORED IN A DATA FILE "QAMRI"        *
C           WHICH IS READ BY THE MFM LINK PROGRAM.                       *
C*****************************************************************************

      INTEGER ALPHA(0:31),MAG(0:31),N
      REAL    RE(0:31),IM(0:31),PI,PHASE(0:31)

      CALL EXCMS('FILEDEF 9 DISK QAM32A DATA (PERM')
      OPEN(6,FILE='QAMRI32A',FORM='FORMATTED',ACCESS='SEQUENTIAL',
     +     STATUS='UNKNOWN')

      N=31
      PI=4.0*ATAN(1.0)

      DO 1 I=0,N
         READ(9,*)ALPHA(I),MAG(I),PHASE(I)
         PHASE(I)=PHASE(I)*PI/180.0
         RE(I)=MAG(I)*COS(PHASE(I))
         IM(I)=MAG(I)*SIN(PHASE(I))
         WRITE(6,*)RE(I),IM(I),MAG(I)/1.0
1     CONTINUE
      ENDFILE(6)
      CLOSE(6)
      END
```

APPENDIX C. SIMULATION PROGRAM: MFMLINK

```
C*****************************************************************************
C*****************************************************************************
C*                                                                           **
C*      MFM COMMUNICATION LINK                                               **
C*                                                                           **
C*                                                                           **
C*        -THIS SIMULATES AN MFM COMMUNICATION LINK WHICH IS                 **
C*           CORRUPTED BY AWGN.  IN DEALING WITH GALOIS FIELDS               **
C*           IT WAS NECESSARY TO TREAT THE ELEMENTS IN TERMS                 **
C*           OF THEIR POWER OF ALPHA PLUS ONE, EXCEPT FOR THE               **
C*           ELEMENT "0" WHICH IS ALWAYS WRITTEN AS "0".                    **
C*                                                                           **
C*****************************************************************************
C*                                                                           **
C*        -THE IMPORTANT ARRAYS/VARIABLES ARE AS FOLLOWS:                    **
C*                                                                           **
C*     ARRAYS: D.....RANDOM DATA VECTOR (CODED CASES)                        **
C*             DIN.....RANDOM DATA VECTOR (UNCODED CASES)                    **
C*             CW.....REED-SOLOMON CODEWORD FOR D                            **
C*             TX.....ANALOG TRANSMISSION (OUTPUT OF INV-FFT)                **
C*             AWGN.....NOISE VECTOR                                         **
C*             RX.....ANALOG RECEPTION (TX + AWGN, OUT OF FFT)               **
C*             R.....RECEIVED VECTOR                                         **
C*             DOUT.....RECEIVED VECTOR (UNCODED CASES)                      **
C*             E.....ESTIMATED ERROR IN R                                    **
C*             V.....ESTIMATED CODEWORD(R + E)                               **
C*             DHAT.....ESTIMATED DATA FROM R                                **
C*             GFA(I,J)....GALOIS FIELD SUM OF SYMBOLS                       **
C*             GFD(I,J)....NUMBER OF BITS THAT I AND J DIFFER IN             **
C*             S.....RECEIVED VECTOR SYNDROME                                **
C*                                                                           **
C*****************************************************************************

      INTEGER B,C,K,BYTERR,ERRORS,FIXED,BDS,RUB,NLESSK,MAX
      INTEGER N,M,ITER,ERASE,DATERR,CCWW(17,0:31),DIN(0:255),DOUT(0:255)
      INTEGER S(-2:2),GFA(0:31,0:31),R(0:31),V(0:31),E(0:31)
      INTEGER TRACE(0:31),T4(0:31),D(0:31),CW(0:31),DD(17,0:31)
      INTEGER GAMMA1,GAMMA2,GAMMA3,ZEROS,TRIERR,DUOERR,UNOERR,OOPS
      INTEGER DHAT(0:31),BDTYPE,CODE,K1,K2,PTS,CPB,PTR,BAUDNO
      INTEGER GFD32(0:31,0:31),DBERR,CWBERR,GFD16(0:15,0:15)
      INTEGER QPDIFF(0:3,0:3)
      REAL D1(0:31),D0(0:255),QAMRE(0:31),QAMIM(0:31),QAMMAG(0:31)
      REAL MEAN,NVAR,AWGN(0:4095),TEMP1(0:4095),TEMP2(0:4095)
      COMPLEX QAMCOM(0:31),TX(0:4095),RX(0:4095),T(0:4095)
      REAL SUM,SUMSQ,SNR,NFIX,QPRE(0:3),QPIM(0:3)

      CALL EXCMS('FILEDEF 9 DISK GFA16 DATA (PERM')
      CALL EXCMS('FILEDEF 7 DISK TR16 DATA (PERM)')
      CALL EXCMS('FILEDEF 8 DISK QAMRI16 FILE (PERM')
      CALL EXCMS('FILEDEF 1 DISK GFD16 FILE (PERM')
      CALL EXCMS('FILEDEF 11 DISK GFA32 FILE (PERM')
```

```
      CALL EXCMS('FILEDEF 12 DISK TR32 FILE (PERM)')
      CALL EXCMS('FILEDEF 13 DISK QAMRI32A FILE (PERM')
      CALL EXCMS('FILEDEF 31 DISK GFD32 FILE (PERM')
      CALL EXCMS('FILEDEF 14 DISK QPSKRI FILE (PERM')
      CALL EXCMS('FILEDEF 34 DISK QPSKDI FILE (PERM')



      OPEN(6,FILE='RSRSLTS',FORM='FORMATTED',ACCESS='SEQUENTIAL',
     +     STATUS='UNKNOWN')
      OPEN(33,FILE='C2C1NEW',FORM='FORMATTED',ACCESS='SEQUENTIAL',
     +     STATUS='UNKNOWN')

C-----------------------------------------------------------------------
C  SELECT WHICH CODE DESIRED...
      WRITE(5,*)' '
      WRITE(5,*)'1. SELECT CODE TYPE...'
 413  WRITE(5,*)'           0 = UNCODED (16-QAM)'
      WRITE(5,*)'           1 = (15,10) REED-SOLOMON CODE (16-QAM)'
      WRITE(5,*)'           2 = (31,26) REED SOLOMON CODE (32-QAM)'
      WRITE(5,*)'           3 = (15,12) REED SOLOMON CODE (16-QAM)'
      WRITE(5,*)'           4 = (31,28) REED SOLOMON CODE (32-QAM)'
      WRITE(5,*)'           5 = UNCODED (32-QAM)'
      WRITE(5,*)'           6 = UNCODED QPSK'
      READ(5,*)CODE

      IF ((CODE.LT.0).OR.(CODE.GT.6)) THEN
          WRITE(5,*)'BAD CHOICE OF CODE--TRY AGAIN?'
          GO TO 413
      ENDIF
      IF (CODE.EQ.6) THEN
         WRITE(5,*)' '
 21      WRITE(5,*)'2. SELECT BAUD TYPE (1-5)'
         READ(5,*)BDTYPE
         IF ((BDTYPE.GT.5).OR.(BDTYPE.LT.1)) THEN
             GO TO 21
         ENDIF
      ENDIF
      IF ((CODE.EQ.1).OR.(CODE.EQ.0).OR.(CODE.EQ.3)) THEN
         N=15
         MAX=15
         NLESSK=5
         IF (CODE.EQ.3) THEN
            MAX=15
            NLESSK=3
         ENDIF
         WRITE(5,*)' '
 2       WRITE(5,*)'2. SELECT BAUD TYPE (1-5)'
         READ(5,*)BDTYPE
         IF ((BDTYPE.GT.5).OR.(BDTYPE.LT.1)) THEN
             GO TO 2
         ENDIF
      ENDIF
      IF (CODE.EQ.6) THEN
         MAX=1
      ENDIF
```

```
      IF ((CODE.EQ.2).OR.(CODE.EQ.4).OR.(CODE.EQ.5)) THEN
         N=31
         MAX=31
         NLESSK=5
         IF (CODE.EQ.4) THEN
            MAX=31
            NLESSK=3
         ENDIF
         WRITE(5,*)' '
 221     WRITE(5,*)'2. SELECT BAUD TYPE (2-5)'
         READ(5,*)BDTYPE
         IF ((BDTYPE.GT.5).OR.(BDTYPE.LT.2)) THEN
             GO TO 221
         ENDIF
      ENDIF
      IF ((CODE.GT.0).AND.(CODE.LT.5)) THEN
         WRITE(5,*)' '
         WRITE(5,*)'3. ERASE CODEWORDS WITH UNCORRECTED ERRORS ?'
         WRITE(5,*)'        ENTER "0" TO LEAVE CODEWORDS AS IS'
         WRITE(5,*)'        ENTER "1" TO ERASE'
         READ(5,*)RUB
      ENDIF

C     WRITE(5,*)' '
C     WRITE(5,*)'4. SELECT SNR(DB,REAL NUMBER)'
C     READ(5,*)SNR

C-----------------------------------------------------------------------
C  READ IN DATA AND CONSTANTS...
C
      CALL BAUDS(CODE,BDTYPE,K1,K2,PTS,CPB,M)
      IF (CODE.EQ.6) THEN
         DO 108 I=0,3
            READ(34,*)(QPDIFF(I,J),J=0,3)
            WRITE(5,*)(QPDIFF(I,J),J=0,3)
            READ(14,*)QPRE(I),QPIM(I)
            QAMCOM(I)=CMPLX(QPRE(I),QPIM(I))

 108     CONTINUE
         GO TO 597
      ENDIF
      CALL DATA(CODE,N,GFA,TRACE,T4,QAMCOM,QAMRE,QAMIM,QAMMAG,GFD16,
     +     GFD32)

597   NFIX=10*LOG10((PTS/2)/(K2-K1)/1.0)

      DO 544 SNR=15.0,22.0,1.0

         BYTERR=0
         NOERR=0
         UNOERR=0
         FUOERR=0
         TRIERR=0
         FIXED=0
         DATERR=0
```

```
          CWBERR=0
          DBERR=0
          ITER=0
          ERASE=0
          MEAN=0.0

        DO 543 BDS=1,100
C-------------------------------------------------------------------------

          DO 15 I=0,MAX-1
                CW(I)=0
                D(I)=0
 15       CONTINUE
          DO 16 I=0,PTS-1
                TX(I)=(0.0,0.0)
                RX(I)=(0.0,0.0)
                T(I)=(0.0,0.0)
 16       CONTINUE
C-------------------------------------------------------------------------
C  MAKE UP RANDOM DATA, ENCODE IT, LOAD IT IN TRANSMITTER...
         IF ((CODE.EQ.0).OR.(CODE.EQ.5)) THEN
            CALL NOCODE(CPB,D0,T,K1,PTS,QAMCOM,DIN,N)
            GO TO 797
         ENDIF
         IF (CODE.EQ.6) THEN
             CALL QPSK(D0,K1,K2,PTS,QAMCOM,DIN,T)
             GO TO 797
         ENDIF
         PTR=K1
         DO 776 J=1,CPB
            CALL RNUN(MAX-NLESSK,D1)
            DO 1 I=0,MAX-NLESSK-1
                 D1(I)=INT((N+1)*D1(I))
 1          CONTINUE
            CALL INTRAN(D1,D,MAX,NLESSK)
            CALL ENCODE(N,CW,D,GFA,CODE,NLESSK,MAX)
            DO 17 I=PTR,PTR+MAX-1
                  T(I)=QAMCOM(CW(I-PTR))
                  T(PTS-I)=CONJG(T(I))
                  DD(J,I-PTR)=D(I-PTR)
                  CCWW(J,I-PTR)=CW(I-PTR)
 17         CONTINUE
            PTR=PTR+MAX
 776     CONTINUE

C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C     TRANSMIT THE CODEWORD...
 797     CALL BITREV(PTS,M,T,TX)
         CALL INVFFT(PTS,M,TX)
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
C  ANY NOISE IN COMMUNICATION CHANNEL? (R(I) IS RECEIVED VECTOR)...
         SUMSQ=0.0
         SUM=0.0
         DO 13 I=0,PTS-1
```

```
            TX(I)=TX(I)/2+CONJG(TX(I))/2
            SUM=SUM+REAL(TX(I))
            SUMSQ=SUMSQ+(REAL(TX(I))**2)
 13      CONTINUE
         NVAR=SUMSQ/(PTS-1)/(10**((SNR-NFIX)/10.0))

         CALL GAUSS(MEAN,NVAR,PTS,AWGN,TEMP1,TEMP2)
         DO 617 I=0,PTS-1
            TX(I)=TX(I)+AWGN(I)
 617     CONTINUE
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C   RECEIVE THE DATA...
         CALL BITREV(PTS,M,TX,RX)
         CALL FFT(PTS,M,RX)
C   DECODE EACH RECEIVED CODEWORD SEQUENTIALLY...
         IF (CODE.EQ.0) THEN
            CALL WHAT16(RX,CPB,PTS,K1,DOUT)
            PTR=K1
            GO TO 667
         ENDIF
         IF (CODE.EQ.5) THEN
            CALL what32(RX,N,PTS,QAMRE,QAMIM,DOUT,K1,CPB)
            PTR=K1
            GO TO 668
         ENDIF
         IF (CODE.EQ.6) THEN
            CALL WHATQP(RX,K1,K2,PTS,DOUT)
            GO TO 669
         ENDIF

         PTR=K1
         DO 777 J=1,CPB
              B=0
              C=0
              K=0
              DO 881 I=0,MAX-1
                     CW(I)=CCWW(J,I)
                     D(I)=DD(J,I)
                     DHAT(I)=0
                     V(I)=0
                     E(I)=0
                     R(I)=0
 881          CONTINUE
              IF ((CODE.EQ.1).OR.(CODE.EQ.3)) THEN
                 CALL WHAT16(RX,MAX,PTS,PTR,R)
              ENDIF
              IF ((CODE.EQ.2).OR.(CODE.EQ.4)) THEN
                 CALL what32(RX,MAX,PTS,QAMRE,QAMIM,R,PTR,MAX)
              ENDIF
              PTR=PTR+MAX
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C   CALCULATE THE SYNDROME...
              ZEROS=0
              NEAREN=0
```

```
              OOPS=0
              CALL SYNDRO(R,S,N,NEAREN,OOPS,GFA,ZEROS,MAX,CODE)
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C       CHECK FOR NO ERRORS...
 199          IF (ZEROS.EQ.NLESSK) THEN
                 NOERR=NOERR+1
                 CALL RUBOUT(RUB,R,V,DHAT,N,NLESSK,ZEROS,MAX)
                 GO TO 112
              ENDIF
C-------------------------------------------------------------------
C-------------------------------------------------------------------

C       GET INTO REST OF DECODING ALGORITHM...
              IF ((CODE.EQ.1).OR.(CODE.EQ.2)) THEN
                 BAUDNO=J
                 B=0
                 C=0
                 K=0
                 CALL DECTWO(R,S,V,DHAT,E,ERRORS,UNOERR,DUOERR,TRIERR
     +              ,FIXED,B,C,K,ERASE,TRACE,N,GFA,GAMMA1,GAMMA2,GAMMA3
     +              ,RUB,NLESSK,ZEROS,CODE,MAX)
              ENDIF
              IF ((CODE.EQ.3).OR.(CODE.EQ.4)) THEN
                 CALL DECONE(S,V,ZEROS,RUB,R,DHAT,N,NLESSK,MAX,GFA,CODE,
     +                       UNOERR,DUOERR)
              ENDIF
C-------------------------------------------------------------------
C   NOW CHECK IF ANY SYMBOL ERRORS WERE MADE (COMPARE ESTIMATE
C         OF CODEWORD SENT TO ACTUAL CODEWORD SENT)...
 112          ERRORS=0
              DO 61 I=0,MAX-1
                 IF (V(I).NE.CW(I)) THEN
                    BYTERR=BYTERR+1
                    ERRORS=1
                     IF ((V(I).EQ.-1).AND.((CODE.EQ.1).OR.(CODE.EQ.3)))
     +                 THEN
                         CWBERR=MAX*4
                         DBERR=(MAX-NLESSK)*4
                         GO TO 388
                     ENDIF
                     IF ((V(I).EQ.-1).AND.((CODE.EQ.2).OR.(CODE.EQ.4)))
     +                 THEN
                         CWBERR=MAX*5
                         DBERR=(MAX-NLESSK)*5
                         GO TO 388
                     ENDIF
                     IF ((CODE.EQ.0).OR.(CODE.EQ.1).OR.(CODE.EQ.3)) THEN
                        CWBERR=CWBERR+GFD16(V(I),CW(I))
                        GO TO 133
                     ENDIF
                     CWBERR=CWBERR+GFD32(V(I),CW(I))
 133              ENDIF
                  ERRORS=1
 61           CONTINUE
              DO 336 I=0,MAX-NLESSK-1
```

```
                IF (D(I).NE.DHAT(I)) THEN
                   DATERR=DATERR+1
                   IF (DHAT(I).EQ.-1) THEN
                      DBERR=DBERR+4
                     ELSE
                   IF ((CODE.EQ.0).OR.(CODE.EQ.1).OR.(CODE.EQ.3)) THEN
                        DBERR=DBERR+GFD16(D(I),DHAT(I))
                        GO TO 231
                   ENDIF
                   DBERR=DBERR+GFD32(D(I),DHAT(I))
 231             ENDIF
                ENDIF
 336        CONTINUE
C-----------------------------------------------------------------------
 388        BAUDNO=J
C           IF (ERRORS.EQ.1) THEN
C113           CALL OUTPUT(S,GAMMA1,GAMMA2,GAMMA3,R,V,B,C,K,CW,D,
C    +                      BAUDNO,DHAT,BDTYPE,MAX)
C           ENDIF
 335        NOP=0
  99        ITER=ITER+1
 777  CONTINUE
 667     IF (CODE.EQ.0) THEN
            DO 942 I=0,CPB-1
               IF (DIN(I).NE.DOUT(I)) THEN
                  DATERR=DATERR+1
                  DBERR=DBERR+GFD16(DIN(I),DOUT(I))
               ENDIF
 942        CONTINUE
         ENDIF
 668     IF (CODE.EQ.5) THEN
            DO 944 I=0,CPB-1
               IF (DIN(I).NE.DOUT(I)) THEN
                  DATERR=DATERR+1
                  DBERR=DBERR+GFD32(DIN(I),DOUT(I))
               ENDIF
 944        CONTINUE
         ENDIF
 669     IF (CODE.EQ.6) THEN
            DO 945 I=0,K2-K1
               IF (DIN(I).NE.DOUT(I)) THEN
                  DATERR=DATERR+1
                  DBERR=DBERR+QPDIFF(DIN(I),DOUT(I))
               ENDIF
 945        CONTINUE
          ENDIF
 543  CONTINUE
      IF (CODE.EQ.0) THEN
         BE=CPB*4*(BDS-1)/1.0
         BE=DBERR/BE
         WRITE(33,*)SNR,BE
      ENDIF
      IF (CODE.EQ.5) THEN
         BE=CPB*5*(BDS-1)/1.0
         BE=DBERR/BE
         WRITE(33,*)SNR,BE
```

```
      ENDIF
      IF (CODE.EQ.6) THEN
         BE=(K2-K1+1)*2*(BDS-1)/1.0
         BE=DBERR/BE
         WRITE(33,*)SNR,BE
      ENDIF
      IF ((CODE.EQ.1).OR.(CODE.EQ.3)) THEN
         BE=CPB*N*4*(BDS-1)/1.0
         BE=DBERR/BE
         WRITE(33,*)SNR,BE
      ENDIF
      IF ((CODE.EQ.2).OR.(CODE.EQ.4)) THEN
         BE=CPB*N*5*(BDS-1)/1.0
         BE=DBERR/BE
         WRITE(33,*)SNR,BE
      ENDIF

 544  CONTINUE
C     WRITE(6,*)'      SNR (DB) =',SNR
C     WRITE(6,*)'     BAUD TYPE = ',BDTYPE
C     IF (CODE.GT.0) THEN
C        IF (CODE.EQ.1) THEN
C           WRITE(6,*)'CODING CHOICE = REED-SOLOMON (15,10)'
C           GO TO 756
C        ENDIF
C        IF (CODE.EQ.2) THEN
C           WRITE(6,*)'CODING CHOICE = REED-SOLOMON (31,26)'
C           GO TO 756
C        ENDIF
C        IF (CODE.EQ.3) THEN
C           WRITE(6,*)'CODING CHOICE = REED-SOLOMON (15,12)'
C           GO TO 756
C        ENDIF
C        IF (CODE.EQ.4) THEN
C           WRITE(6,*)'CODING CHOICE = REED-SOLOMON (31,28)'
C           GO TO 756
C        ENDIF
C756     WRITE(6,*)' CW SYMBOL ERRORS AFTER CORRECTION=',BYTERR
C        WRITE(6,*)'          TOTAL SYMBOLS TRANSMITTED=',N*ITER
C        WRITE(6,*)'     RESULTING CODEWORD BIT ERRORS=',CWBERR
C        IF ((CODE.EQ.1).OR.(CODE.EQ.3)) THEN
C           WRITE(6,*)'          CODEWORD BITS TRANSMITTED=',N*ITER*4
C           BE=N*ITER*4/1.0
C           BE=CWBERR/BE
C               ELSE
C           WRITE(6,*)'          CODEWORD BITS TRANSMITTED=',N*ITER*5
C           BE=N*ITER*5/1.0
C           BE=CWBERR/BE
C        ENDIF
C
C           WRITE(6,*)'           CODEWORDS WITH NO ERRORS=',NOERR
C           WRITE(6,*)'             DETECTED SINGLE ERRORS=',UNOERR
C           WRITE(6,*)'             DETECTED DOUBLE ERRORS=',DUOERR
C           IF ((CODE.EQ.1).OR.(CODE.EQ.2)) THEN
C              WRITE(6,*)'  DETECTED TRIPLE (OR MORE) ERRORS=',TRIERR
C              WRITE(6,*)'   CORRECTED TRIPLE (BURST) ERRORS=',FIXED
```

```
C           ENDIF
C           WRITE(6,*)'                              ERASURES=',ERASE
C           WRITE(6,*)'       TOTAL CODEWORDS TRANSMITTED=',ITER
C      ENDIF
C      IF (CODE.EQ.0) THEN
C         WRITE(6,*)'CODING CHOICE = UNCODED (QAM-16)'
C         WRITE(6,*)'                  DATA SYMBOL ERRORS=',DATERR
C         WRITE(6,*)'          DATA SYMBOLS TRANSMITTED=',CPB*(BDS-1)
C         WRITE(6,*)'                    DATA BIT ERRORS=',DBERR
C         WRITE(6,*)'              DATA BITS TRANSMITTED=',CPB*4*(BDS-1)
C      ENDIF
C      IF (CODE.EQ.5) THEN
C         WRITE(6,*)'CODING CHOICE = UNCODED (QAM-32)'
C         WRITE(6,*)'                  DATA SYMBOL ERRORS=',DATERR
C         WRITE(6,*)'          DATA SYMBOLS TRANSMITTED=',CPB*(BDS-1)
C         WRITE(6,*)'                    DATA BIT ERRORS=',DBERR
C         WRITE(6,*)'              DATA BITS TRANSMITTED=',CPB*5*(BDS-1)
C      ENDIF
C      WRITE(6,*)' '
C      WRITE(6,*)'BIT ERROR RATE = ',BE
C      WRITE(6,*)'               TOTAL BAUDS EXAMINED=',BDS-1


       WRITE(33,*)'TOTAL BITS IN EACH SNR=',(K2-K1)*4*(BDS-1)
       ENDFILE(6)
       CLOSE(6)
       ENDFILE(33)
       CLOSE(33)
       END
C*******************************************************************************************

       SUBROUTINE BAUDS(CODE,BDTYPE,K1,K2,PTS,CPB,M)

C    SUBROUTINE TO SET SOME INITIAL VALUES, DEPENDING ON BAUD TYPE...
C
C      CONSTANTS:  PTS.....FFT SIZE
C                  M.......POWER OF 2 (OF PTS)
C                  K1......FIRST TONE NUMBER FOR 16-20KHZ BAND WITH
C                                SAMPLING FREQ=61440HZ AND FFT SIZE PTS
C                  K2......LAST TONE FOR THIS BAND
C                  CPB.....CODEWORDS PER BAUD (ONE (15,10) REED-SOLOMON
C                                CODEWORD PER 15 TONES IN THIS BAND)
C
       INTEGER BDTYPE,K1,K2,PTS,CPB,M,CODE

       IF (BDTYPE.EQ.1) THEN
          PTS=256
          M=8
          CPB=1
          K1=68
          K2=83
        ENDIF
        IF (BDTYPE.EQ.2) THEN
          PTS=512
          M=9
          CPB=2
```

```
        K1=135
        K2=166
      ENDIF
      IF (BDTYPE.EQ.3) THEN
        PTS=1024
        M=10
        CPB=4
        K1=269
        K2=332
      ENDIF
      IF (BDTYPE.EQ.4) THEN
        PTS=2048
        M=11
        CPB=8
        K1=537
        K2=664
      ENDIF
      IF (BDTYPE.EQ.5) THEN
        PTS=4096
        M=12
        IF (CODE.EQ.1) THEN
           CPB=17
         ELSE
           CPB=16
        ENDIF
        K1=1073
        K2=1328
      ENDIF
      IF ((CODE.EQ.0).OR.(CODE.GE.5)) THEN
          CPB=K2-K1+1
      ENDIF
      IF ((CODE.EQ.2).OR.(CODE.EQ.4)) THEN
         CPB=INT(CPB/2)
      ENDIF
      RETURN
      END

C*******************************************************************************
      SUBROUTINE DATA(CODE,N,GFA,TRACE,T4,QAMCOM,QAMRE,QAMIM,QAMMAG,
     +    GFD16,GFD32)

C  THIS SUBROUTINE PROVIDES DATA TO THE PROGRAM DEPENDING ON THE CHOICE
C     OF CODE.
C
C       ROWS.....NUMBER OF ROWS IN DATA FILE FOR EACH ROW IN GF
C                ADDITION TABLE (16 ENTRIES IN EACH ROW OF DATA FILE)
C       QAMRE.....REAL PART OF A QAM CONSTELLATION ELEMENT
C       QAMIM.....IMAGINARY PART OF A QAM CONSTELLATION ELEMENT
C       QAMCOM....THE COMPLEX ARRAY OF (QAMRE,QAMRI)
C       QAMMAG....MAGNITUDE OF THAT QAM ELEMENT
C
C-------------------------------------------------------------------
      INTEGER N,SUM,ROWS,CODE
      INTEGER GFA(0:N,0:N),GFD16(0:15,0:15),TRACE(0:N),T4(0:N)
      INTEGER GFD32(0:31,0:31)
      REAL    QAMRE(0:N),QAMIM(0:N),X,Y,QAMMAG(0:N)
```

```
      COMPLEX QAMCOM(0:N)

      IF (CODE.EQ.1) THEN
          READ(7,*)(TRACE(I),I=0,N)
          READ(7,*)(T4(I),I=0,N)
      ENDIF
      IF (CODE.EQ.2) THEN
          READ(12,*)(TRACE(I),I=0,N)
      ENDIF
      ROWS=INT(N+1)/16
      DO 10 I=0,N
         SUM=0
         DO 20 J=1,ROWS
            IF ((CODE.EQ.0).OR.(CODE.EQ.1).OR.(CODE.EQ.3)) THEN
               READ(9,*)(GFA(I,K),K=0,15)
               READ(1,*)(GFD16(I,K),K=0,15)
               GO TO 20
            ENDIF
            READ(11,*) (GFA(I,K),K=SUM,SUM+15)
            READ(31,*) (GFD32(I,K),K=SUM,SUM+15)
            SUM=SUM+16
20       CONTINUE
10    CONTINUE
      DO 40 I=0,N
            IF ((CODE.EQ.0).OR.(CODE.EQ.1).OR.(CODE.EQ.3)) THEN
                READ(8,*)QAMRE(I),QAMIM(I),QAMMAG(I)
                GO TO 35
            ENDIF
            READ(13,*)QAMRE(I),QAMIM(I),QAMMAG(I)
35          X=QAMRE(I)
            Y=QAMIM(I)
            QAMCOM(I)=CMPLX(X,Y)
40    CONTINUE
      RETURN
      END
C*******************************************************************************************
      SUBROUTINE NOCODE(CPB,D0,T,K1,PTS,QAMCOM,DIN,N)

C  THIS SUBROUTINE GIVES RANDOM DATA FOR THE UNCODED CASES AND
C       LOADS IT INTO THE TRANSMITTER...
C
C     DIN.......INTEGER REPRESENTING AN ELEMENT OF GF(2**M)
C     D0........REAL NUMBER EQUAL TO INTEGER DIN (FOR INV-FFT)
C     T.........CONJUGATE SYMMETRIC SPECTRUM FOR INV-FFT
C
C-------------------------------------------------------------------

      INTEGER CPB,K1,DIN(0:255),PTS,N
      REAL    D0(0:255)
      COMPLEX T(0:4095),QAMCOM(0:31)

      CALL RNUN(CPB,D0)
      DO 10 I=0,CPB-1
            D0(I)=INT((N+1)*D0(I))
            DO 20 J=0,N
               IF (D0(I).EQ.J/1.0) THEN
```

```
                  DIN(I)=J
               ENDIF
 20          CONTINUE
             T(K1+I)=QAMCOM(DIN(I))
             T(PTS-K1-I)=CONJG(T(K1+I))
 10    CONTINUE
       RETURN
       END


C*******************************************************************************************
       SUBROUTINE QPSK(D0,K1,K2,PTS,QAMCOM,DIN,T)

C  THIS SUBROUTINE GIVES RANDOM DATA FOR UNCODED QPSK AND
C       LOADS IT INTO THE TRANSMITTER...
C
C      DIN.......INTEGER REPRESENTING AN ELEMENT OF GF(2**M)
C      D0........REAL NUMBER EQUAL TO INTEGER DIN (FOR INV-FFT)
C      T.........CONJUGATE SYMMETRIC SPECTRUM FOR INV-FFT
C
C-------------------------------------------------------------------

       INTEGER K1,K2,DIN(0:255),PTS
       REAL    D0(0:255)
       COMPLEX T(0:4095),QAMCOM(0:31)

       CALL RNUN(K2-K1+1,D0)
       DO 10 I=0,K2-K1
             D0(I)=INT(4*D0(I))
             DO 20 J=0,3
                IF (D0(I).EQ.J/1.0) THEN
                   DIN(I)=J
                ENDIF
 20          CONTINUE
             T(K1+I)=QAMCOM(DIN(I))
             T(PTS-K1-I)=CONJG(T(K1+I))
 10    CONTINUE
       RETURN
       END


C*******************************************************************************************
       SUBROUTINE ENCODE(N,CW,D,GFA,CODE,NLESSK,MAX)

C    THIS SUBROUTINE IS THE REED-SOLOMON ENCODER (SYSTEMATIC FORM).
C
C      G0-G4.....COEFFICIENTS OF GENERATOR POLYNOMIAL
C      SR........SHIFT REGISTERS
C
C-------------------------------------------------------------------

       INTEGER N,INPUT,G0,G1,G2,G3,G4,CODE,NLESSK,MAX
       INTEGER CW(0:MAX-1),D(0:MAX-1),GFA(0:N,0:N),SR(5)

C  SET GENERATOR POLYNOMIAL CONSTANTS...
       G1=5
```

```
      G2=12
      G3=12
      G4=5
      IF (CODE.EQ.2) THEN
         G1=14
         G2=18
         G3=18
         G4=14
      ENDIF
      IF (CODE.EQ.3) THEN
         G0=4
         G1=12
         G2=11
      ENDIF
      IF (CODE.EQ.4) THEN
         G0=4
         G1=13
         G2=12
      ENDIF
C   ENCODE THE DATA FOR THE SYSTEMATIC (N,N-5) CODE...

 411  DO 8 J=1,NLESSK
         SR(J)=0
 8    CONTINUE
      DO 9 J=0,MAX-NLESSK-1
           CW(MAX-1-J)=D(MAX-NLESSK-1-J)
           INPUT=GFA(SR(1),D(MAX-NLESSK-1-J))
           IF (INPUT.EQ.0) THEN
               DO 40 K=1,NLESSK-1
                    SR(K)=SR(K+1)
 40            CONTINUE
               SR(NLESSK)=INPUT
               GO TO 9
           ENDIF
           IF (NLESSK.EQ.5) THEN
               SR(1)=GFA(SR(2),MOD(INPUT+G4-2,N)+1)
               SR(2)=GFA(SR(3),MOD(INPUT+G3-2,N)+1)
               SR(3)=GFA(SR(4),MOD(INPUT+G2-2,N)+1)
               SR(4)=GFA(SR(5),MOD(INPUT+G1-2,N)+1)
               SR(5)=INPUT
           ENDIF
           IF (NLESSK.EQ.3) THEN
               SR(1)=GFA(SR(2),MOD(INPUT+G2-2,N)+1)
               SR(2)=GFA(SR(3),MOD(INPUT+G1-2,N)+1)
               SR(3)=MOD(INPUT+G0-2,N)+1
           ENDIF
 9    CONTINUE
      DO 11 I=0,NLESSK-1
         CW(I)=SR(NLESSK-I)
 11   CONTINUE
      RETURN
      END
C*******************************************************************************
      SUBROUTINE WHAT16(X,MAX,L,PTR,R)
```

```
C  SUBROUTINE TO DECIDE WHICH ELEMENT OF GF(N+1) EACH ELEMENT
C      OF THE RECEIVED DATA CORRESPONDS TO.
C
C         PTR.....CURRENT CODEWORD STARTING TONE
C         X().....AN ELEMENT FROM THE RECEIVER (FFT)
C         MAX.....CODEWORD BLOCK LENGTH
C         L.......FFT SIZE
C         R().....THE RECEIVED CODEWORD IN TERMS OF ELEMENTS OF GF(N+1)
C
C-----------------------------------------------------------------------

      INTEGER MAX,R(0:MAX-1),PTR
      COMPLEX X(0:L-1)
      REAL    A,B,Z,MID

      MID=150.0
      DO 10 J=PTR,PTR+MAX-1
            A=REAL(X(J))
            B=AIMAG(X(J))
            Z=CABS(X(J))
            IF ((A.GE.0).AND.(B.GE.0)) THEN
               IF ((Z.GE.MID).AND.(A.GE.B)) THEN
                   R(J-PTR)=4
                   GO TO 10
               ENDIF
               IF ((Z.GE.MID).AND.(A.LT.B)) THEN
                   R(J-PTR)=7
                   GO TO 10
               ENDIF
               IF ((Z.LT.MID).AND.(A.GE.B)) THEN
                   R(J-PTR)=0
                   GO TO 10
               ENDIF
               R(J-PTR)=3
               GO TO 10
            ENDIF
            IF ((A.GE.0).AND.(B.LT.0)) THEN
               IF ((Z.GE.MID).AND.(A.GE.ABS(B))) THEN
                   R(J-PTR)=15
                   GO TO 10
               ENDIF
               IF ((Z.GE.MID).AND.(A.LT.ABS(B))) THEN
                   R(J-PTR)=14
                   GO TO 10
               ENDIF
               IF ((Z.LT.MID).AND.(A.GE.ABS(B))) THEN
                   R(J-PTR)=1
                   GO TO 10
               ENDIF
               R(J-PTR)=9
               GO TO 10
            ENDIF
            IF ((A.LT.0).AND.(B.LT.0)) THEN
               IF ((Z.GE.MID).AND.(ABS(A).GE.ABS(B))) THEN
                   R(J-PTR)=8
                   GO TO 10
```

```
                ENDIF
                IF ((Z.GE.MID).AND.(ABS(A).LT.ABS(B))) THEN
                    R(J-PTR)=13
                    GO TO 10
                ENDIF
                IF ((Z.LT.MID).AND.(ABS(A).GE.ABS(B))) THEN
                    R(J-PTR)=5
                    GO TO 10
                ENDIF
                R(J-PTR)=11
                GO TO 10
             ENDIF
             IF ((A.LT.0).AND.(B.GE.0)) THEN
                IF ((Z.GE.MID).AND.(ABS(A).GE.B)) THEN
                    R(J-PTR)=10
                    GO TO 10
                ENDIF
                IF ((Z.GE.MID).AND.(ABS(A).LT.B)) THEN
                    R(J-PTR)=12
                    GO TO 10
                ENDIF
                IF ((Z.LT.MID).AND.(ABS(A).GE.B)) THEN
                    R(J-PTR)=2
                    GO TO 10
                ENDIF
                R(J-PTR)=6
             ENDIF
 10    CONTINUE
             RETURN
             END
C*****************************************************************************************
      SUBROUTINE WHATQP(X,K1,K2,L,R)

C  SUBROUTINE TO DECIDE WHICH ELEMENT OF GF(N+1) EACH ELEMENT
C       OF THE RECEIVED DATA CORRESPONDS TO.
C
C          X().....AN ELEMENT FROM THE RECEIVER (FFT)
C          K1.....FIRST TONE IN BAUD
C          K2.....SECOND TONE IN BAUD
C          L.....FFT SIZE
C          R().....THE RECEIVED CODEWORD IN TERMS OF ELEMENTS OF GF(N+1)
C
C-------------------------------------------------------------------

      INTEGER R(0:K2-K1+1)
      COMPLEX X(0:L-1)
      REAL    A,B

      DO 10 J=K1,K2
            A=REAL(X(J))
            B=AIMAG(X(J))
            IF ((A.GE.0).AND.(B.GE.0)) THEN
                R(J-K1)=0
                GO TO 10
            ENDIF
            IF ((A.LE.0).AND.(B.GE.0)) THEN
```

```
                R(J-K1)=1
                GO TO 10
            ENDIF
            IF ((A.LE.0).AND.(B.LE.0)) THEN
                R(J-K1)=2
                GO TO 10
            ENDIF
            R(J-K1)=3
 10     CONTINUE
        RETURN
        END

C*******************************************************************************
        SUBROUTINE WHAT32(X,N,L,QAMRE,QAMIM,R,PTR,MAX)

C   SUBROUTINE TO FIND CLOSEST QAM CONSTELLATION ELEMENT BY SUBTRACTING
C       THE MAGNITUDE OF THE RECEIVED COMPLEX SIGNAL FROM THE PRODUCT OF
C       THE COMPLEX CONJUGATE OF THE RECEIVED SIGNAL AND EACH ELEMENT OF
C       THE QAM CONSTELLATION, AND TAKING THAT CONSTELLATION ELEMENT
C       WITH THE SMALLEST RESULT AS AN ESTIMATE OF THE RECEIVED SIGNAL.
C       (ONLY USED WITH 32-QAM)
C
C-------------------------------------------------------------------------

        INTEGER N,L,R(0:MAX-1),WHO,PTR
        COMPLEX X(0:L-1)
        REAL    NEW,A,NEWR,NEWI,QAMRE(0:N),QAMIM(0:N)

        DO 10 I=PTR,PTR+MAX-1
              NEWR=ABS(QAMRE(0)-REAL(X(I)))**2
              NEWI=ABS(QAMIM(0)-AIMAG(X(I)))**2
              NEW=SQRT(NEWR+NEWI)
              WHO=0
              DO 20 J=1,N
                 A=ABS(QAMRE(J)-REAL(X(I)))**2
                 A=A + ABS(QAMIM(J)-AIMAG(X(I)))**2
                 IF (SQRT(A).LT.NEW) THEN
                    NEW= A
                    WHO=J
                 ENDIF
 20           CONTINUE
              R(I-PTR)=WHO
 10     CONTINUE
        RETURN
        END


C*******************************************************************************
        SUBROUTINE SYNDRO(R,S,N,NEAREN,OOPS,GFA,ZEROS,MAX,CODE)

C   THIS SUBROUTINE COMPUTES THE SYNDROME VECTOR (FOR DECODING)
C
C       S.........SYNDROME VECTOR
C       ZEROS.....NUMBER OF SYNDROME ELEMENTS EQUAL TO ZERO
C       NEAREN....FLAG FROM DECODING SUBROUTINE, =1 IF THIS IS THE
C                 SECOND SYNDROME COMPUTATION FOR A GIVEN CODEWORD
```

```
C                (RECHECKING CORRECTIONS MADE FOR HIGHER ORDER ERRORS)
C     OOPS.......FLAG TO DECODING SYNDROME THAT AN ELEMENT OF SYNDROME
C                IS NOT EQUAL TO ZERO (WHILE PERFORMING SECOND SYNDROME
C                COMPUTATION)
C     LOW........FIRST ELEMENT OF SYNDROME (DIFFERS BETWEEN LENGTH 31
C                AND LENGTH 15 CODES)
C
C-----------------------------------------------------------------------

      INTEGER N,ZEROS,NEAREN,OOPS,X,ALPHA,TEMP,SUM,CODE
      INTEGER R(0:MAX-1),S(-2:2),GFA(0:N,0:N),LOW
      OOPS=0
      DO 5 I=-2,2
           S(I)=0
 5    CONTINUE
      ZEROS=0
      IF (CODE.GT.2) THEN
         LOW=0
        ELSE
         LOW=-2
      ENDIF
      DO 20 ALPHA=LOW,2
            SUM=0
            DO 30 I=0,MAX-1
                  IF (R(I).EQ.0) THEN
                     GO TO 30
                  ENDIF
                  TEMP=0
                  IF (ALPHA.LT.0) THEN
                     TEMP=ALPHA+N
                     X=R(I)-1+TEMP*I
                  ELSE
                     X=R(I)-1+ALPHA*I
                  ENDIF
                  X=MOD(X,N)
                  X=X+1
                  SUM=GFA(SUM,X)
 30         CONTINUE
            S(ALPHA)=SUM
            IF ((SUM.NE.0).AND.(NEAREN.EQ.1)) THEN
               OOPS=1
               GO TO 21
            ENDIF
            IF (S(ALPHA).EQ.0) THEN
               ZEROS=ZEROS+1
            ENDIF
 20   CONTINUE
 21   RETURN
      END

C***********************************************************************************************
      SUBROUTINE DECTWO(R,S,V,DHAT,E,ERRORS,UNOERR,DUOERR,TRIERR
     +    ,FIXED,B,C,K,ERASE,TRACE,N,GFA,GAMMA1,GAMMA2,GAMMA3
     +    ,RUB,NLESSK,ZEROS,CODE,MAX)
C
C   DECODING SUBROUTINE FOR DEC/TED CODES
```

```
C
C        R.............RECEIVED CODEWORD VECTOR
C        S.............SYNDROME VECTOR
C        E.............ESTIMATED ERROR VECTOR
C        V.............ESTIMATED CODEWORD VECTOR
C        DHAT..........ESTIMATED DATA VECTOR (FROM V)
C        ERRORS........FLAG FOR PARAMETERS OUTPUT IN CASE THERE ARE
C                      DECODING ERRORS (ERRORS=1)
C        UNOERR........SINGLE ERRORS
C        DUOERR........DOUBLE ERRORS
C        TRIERR........UNCORRECTABLE TRIPLE ERRORS
C        FIXED.........CORRECTED TRIPLE BURST ERRORS
C        B,C,K.........PARAMETERS USED IN DOUBLE ERROR CORRECTION
C        GAMMA1-3......PARAMETERS USED IN DOUBLE ERROR CORRECTION
C        TRACE.........TRACE OF AN ELEMENT OF GF(2**M)
C        ERASE.........NUMBER OF ERASURES (IF ERASING TRIPLE-ERRORS IS
C                      CHOSEN (IF RUB=1)
C
C-----------------------------------------------------------------------

      INTEGER GAMMA1,GAMMA2,GAMMA3,UNOERR,DUOERR,TRIERR,N,MAX
      INTEGER FIXED,NOP,B,C,K,W1,W2,W3,X1,X2,X3,Y1,Y2,Y3,TEST
      INTEGER ROOT(2),TRACE(0:N),GFA(0:N,0:N)
      INTEGER R(0:N-1),S(-2:2),V(0:N-1),DHAT(0:N-1),E(0:N-1)
      INTEGER Z3,W,X,Y,ERASE,A1,A2,ERRORS,OOPS,ZEROS,RUB,CODE


C-----------------------------------------------------------------------
C      CHECK FOR 3 OR MORE ERRORS FROM SYNDROME...
      IF (ZEROS.GT.2) THEN
         TRIERR=TRIERR+1
         CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
         GO TO 333
      ENDIF

 36   NOP=0
      IF (S(1)*S(-2).EQ.0)  THEN
         A1=0
        ELSE
         A1=MOD(S(1)+S(-2)-2,N)+1
      ENDIF
      IF (S(-1)*S(0).EQ.0) THEN
         A2=0
        ELSE
         A2=MOD(S(-1)+S(0)-2,N)+1
      ENDIF
      GAMMA1=GFA(A1,A2)

      IF ((S(2)*S(-2)).EQ.0) THEN
         A1=0
        ELSE
         A1=MOD(S(2)+S(-2)-2,N)+1
      ENDIF
      IF (S(0).EQ.0) THEN
         A2=0
        ELSE
```

```
         A2=MOD(S(0)*2-2,N)+1
       ENDIF
       GAMMA2=GFA(A1,A2)

       IF ((S(0)*S(1)).EQ.0) THEN
          A1=0
        ELSE
          A1=MOD(S(0)+S(1)-2,N)+1
       ENDIF
       IF ((S(2)*S(-1)).EQ.0) THEN
          A2=0
        ELSE
          A2=MOD(S(2)+S(-1)-2,N)+1
       ENDIF
       GAMMA3=GFA(A1,A2)

C
C-------------------------------------------------------------------
C  STEP#4: CHECK FOR A SINGLE ERROR...

       IF ((GAMMA1.EQ.0).AND.(GAMMA2.EQ.0).AND.(GAMMA3.EQ.0)) THEN
          Y=MOD(S(1)+N-S(0),N)
          E(Y)=S(0)
          UNOERR=UNOERR+1
          GO TO 59
       ENDIF

C-------------------------------------------------------------------
C  STEP#5: CHECK FOR 3 OR MORE ERRORS...

       IF (((GAMMA1.NE.0).OR.(GAMMA2.NE.0).OR.(GAMMA3.NE.0)).AND.(GAMMA1
      +       *GAMMA2*GAMMA3.EQ.0)) THEN
          TRIERR=TRIERR+1
          ERASE=ERASE+1
          CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
          GO TO 333
       ENDIF

C-------------------------------------------------------------------
C  STEP#6: CALCULATE CONSTANTS IN QUADRATIC FORMULA...

       IF (GAMMA2.EQ.0) THEN
          B=0
        ELSE
          B=MOD(GAMMA2+N-GAMMA1,N)+1
       ENDIF
       IF (GAMMA3.EQ.0) THEN
          C=0
        ELSE
          C=MOD(GAMMA3+N-GAMMA1,N)+1
       ENDIF
       IF (C.EQ.0) THEN
          K=0
          GO TO 37
       ENDIF
       IF (B.EQ.1) THEN
```

```
            K=C
            GO TO 37
         ENDIF
         K=MOD(C-1+3*N-2*(B-1),N)+1
 37      NOP=0
         IF ((TRACE(K).EQ.1).AND.(K.NE.6)) THEN
            TRIERR=TRIERR+1
            ERASE=ERASE+1
            CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
            GO TO 333
         ENDIF

C--------------------------------------------------------------------
C    LOOK FOR A CORRECTABLE TRIPLE BURST ERROR...

         IF (K.EQ.6) THEN
            W1=S(-1)-1
            W2=S(1)-1
            W3=MOD(W1+W2,N)
            X1=S(-2)-1
            X2=S(2)-1
            X3=MOD(X1+X2,N)
            Y1=S(0)-1
            IF (N.EQ.15) THEN
               Y2=MOD(2*Y1+3,N)
               Y3=MOD(2*Y1+6,N)
             ELSE
               Y2=MOD(2*Y1+20,N)
               Y3=MOD(2*Y1+23,N)
            ENDIF
         IF ((W3.EQ.Y2).AND.(X3.EQ.Y3)) THEN
            FIXED=FIXED+1
            IF (N.EQ.15) THEN
               Z3=MOD(W2-Y1+3*N-10,N)
             ELSE
               Z3=MOD(W2-Y1+3*N-11,N)
            ENDIF
            E(Z3)=S(0)
            E(Z3+1)=S(0)
            E(Z3+2)=S(0)
            GO TO 59
         ENDIF

         ENDIF

C--------------------------------------------------------------------
C  FIND THE ERRORS USING THE QUADRATIC FORMULA...

         TEST=0
         ROOT(1)=0
         ROOT(2)=0
         DO 50 I=1,N
            W=MOD(2*(I-1),N)+1
            IF (B.EQ.1) THEN
               X=I
               GO TO 41
```

```
          ENDIF
          IF (B.EQ.0) THEN
             X=0
             GO TO 41
          ENDIF
          X=MOD(B+I-2+N,N)+1
 41       Y=GFA(W,X)
          Z=GFA(Y,C)
          IF ((Z.EQ.0).AND.(TEST.EQ.0)) THEN
             ROOT(1)=I-1
 42          TEST=1
             GO TO 50
          ENDIF
          IF ((Z.EQ.0).AND.(TEST.EQ.1)) THEN
             ROOT(2)=I-1
  43         GO TO 55
          ENDIF
 50    CONTINUE


 55    NOP=0


C    ROOT(1) AND ROOT(2) GIVE DOUBLE ERROR LOCATIONS, NOW FIND MAGNITUDE
       IF (S(0).EQ.0) THEN
          A1=0
        ELSE
          A1=MOD(S(0)-1+ROOT(2),N)+1
       ENDIF
       IF (S(1).EQ.0) THEN
          A2=A1
        ELSE
          A2=GFA(A1,S(1))
       ENDIF
       X=MOD(A2+N-B,N)+1

       E(ROOT(1))=X
       E(ROOT(2))=GFA(S(0),X)
       IF (E(ROOT(1))*E(ROOT(2)).EQ.0) THEN
          TRIERR=TRIERR+1
          ERASE=ERASE+1
          CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
          GO TO 333
       ENDIF
       DUOERR=DUOERR+1
 59    NOP=0
C-------------------------------------------------------------------------
C STEP#7: NOW ADD ESTIMATED ERROR VECTOR TO RECEIVED VECTOR...

       ERRORS=0
 72    DO 60 I=0,N-1
          V(I)=GFA(R(I),E(I))
          IF (I.GT.4) THEN
             DHAT(I-5)=V(I)
          ENDIF
 60    CONTINUE
```

```
C-----------------------------------------------------------------------
C  STEP#8: MAKE LAST CHECK FOR 3 OR MORE ERRORS...

      NEAREN=1
      ZEROS=0
      OOPS=0
      CALL SYNDRO(V,S,N,NEAREN,OOPS,GFA,ZEROS,MAX,CODE)
      IF (OOPS.EQ.1) THEN
         TRIERR=TRIERR+1
         ERASE=ERASE+1
         DUOERR=DUOERR-1
         CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
         GO TO 333
      ENDIF

 333  RETURN
      END
C*******************************************************************************
      SUBROUTINE DECONE(S,V,ZEROS,RUB,R,DHAT,N,NLESSK,MAX,GFA,CODE,
     +                  UNOERR,DUOERR)
C
C   DECODING SUBROUTINE FOR SEC/DED CODES
C
C        R.............RECEIVED CODEWORD VECTOR
C        S.............SYNDROME VECTOR
C        V.............ESTIMATED CODEWORD VECTOR
C        DHAT..........ESTIMATED DATA VECTOR (FROM V)
C                      DECODING ERRORS (ERRORS=1)
C        UNOERR........SINGLE ERRORS
C        DUOERR........DOUBLE ERRORS
C                      CHOSEN (IF RUB=1)
C
C-----------------------------------------------------------------------
      INTEGER ZEROS,N,DUOERR,X,Y,UNOERR,MAX,CODE,RUB,NLESSK
      INTEGER S(-2  ),GFA(0:N,0:N),V(0:MAX-1),DHAT(0:MAX-1),R(0:MAX-1)
      INTEGER NEAREN,OOPS

      IF (ZEROS.EQ.1) THEN
          DUOERR=DUOERR+1
          CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
          GO TO 150
      ENDIF
      S(-2)=0
      S(-1)=0
      DO 10 I=0,MAX-1
            V(I)=R(I)
 10   CONTINUE
      IF ((S(0).NE.0).AND.(ZEROS.EQ.2)) THEN
          V(0)=GFA(S(0),R(0))
          UNOERR=UNOERR+1
          GO TO 100
      ENDIF
      IF ((S(1).NE.0).AND.(ZEROS.EQ.2)) THEN
          V(1)=GFA(S(1),R(1))
          UNOERR=UNOERR+1
```

```
            GO TO 100
         ENDIF
         IF ((S(2).NE.0).AND.(ZEROS.EQ.2)) THEN
            V(2)=GFA(S(2),R(2))
            UNOERR=UNOERR+1
            GO TO 100
         ENDIF
         X=MOD(S(1)+N-S(0),N)
         Y=MOD(S(2)+N-S(1),N)
         IF (X.EQ.Y) THEN
C           X=X+3
            V(X)=GFA(R(X),S(0))
            UNOERR=UNOERR+1
            GO TO 100
          ELSE
            DUOERR=DUOERR+1
            CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
            GO TO 150
         ENDIF
C  LAST CHANCE TO CHECK FOR 2 OR MORE ERRORS...
 100     NEAREN=1
         ZEROS=0
         OOPS=0
         CALL SYNDRO(V,S,N,NEAREN,OOPS,GFA,ZEROS,MAX,CODE)
         IF (OOPS.EQ.1) THEN
            DUOERR=DUOERR+1
            UNOERR=UNOERR-1
            CALL RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)
            GO TO 150
         ENDIF
         DO 20 I=3,MAX-1
               DHAT(I-3)=V(I)
 20      CONTINUE
 150     RETURN
         END


C*************************************************************************************************
         SUBROUTINE RUBOUT(RUB,R,V,DHAT,NLESSK,ZEROS,MAX)

C  THIS SUBROUTINE ERASES CODEWORDS WITH MORE ERRORS THAN A GIVEN CODE
C          CAN CORRECT.
C
C-------------------------------------------------------------------

         INTEGER NLESSK,ZEROS,MAX
         INTEGER RUB,R(0:MAX-1),V(0:MAX-1),DHAT(0:MAX-1)

         IF ((RUB.EQ.0).OR.(ZEROS.EQ.NLESSK)) THEN
            DO 14 I=0,MAX-1
                  V(I)=R(I)
                  IF (I.GT.NLESSK-1) THEN
                     DHAT(I-NLESSK)=R(I)
                  ENDIF
 14         CONTINUE
            GO TO 50
```

```
      ENDIF
      IF (RUB.EQ.1) THEN
         DO 101 I=0,MAX-1
            V(I)=-1
            IF (I.LE.MAX-NLESSK-1) THEN
               DHAT(I)=-1
            ENDIF
 101  CONTINUE
      ENDIF
 50   RETURN
      END


C*******************************************************************************

      SUBROUTINE OUTPUT(S,GAMMA1,GAMMA2,GAMMA3,R,V,B,C,K,CW,D,
     +                  BAUDNO,DHAT,BDTYPE,MAX)

C   FOR BYTE ERRORS, WRITE PERTINENT DATA TO RSOUTPUT FILE...
C
C-------------------------------------------------------------------

            INTEGER GAMMA1,GAMMA2,GAMMA3,B,C,K,MAX
            INTEGER S(-2:2),R(0:MAX),V(0:MAX),CW(0:MAX)
            INTEGER D(0:MAX),BAUDNO,DHAT(0:MAX),BDTYPE

            WRITE(6,*)'-------------------------------------------------
     +                 -------------'
            WRITE(6,*)'BAUDTYPE=',BDTYPE
            WRITE(6,*)'CODEWORD NUMBER=',BAUDNO
            WRITE(6,904)
 904        FORMAT('SYNDROMES')
            WRITE(6,905) S(-2),S(-1),S(0),S(1),S(2)
 905        FORMAT(1X,I2,3X,I2,3X,I2,3X,I2,3X,I2)
            WRITE(6,*)' '
            WRITE(6,*)' '
            WRITE(6,906)
 906        FORMAT('GAMMAS')
            WRITE(6,907) GAMMA1,GAMMA2,GAMMA3
 907        FORMAT(5X,I2,5X,I2,5X,I2)
            WRITE(6,*)'B,C,K'
            WRITE(6,*)B,C,K
            WRITE(6,*)' '
            WRITE(6,908)
 908        FORMAT(11X,'I',10X,'DATA',7X,'ENCODE',7X,'RECD',8X,'ESTCW',
     +          5X,'ESTDATA')
            DO 71 J=0,MAX-1
                 WRITE (6,*) J,D(J),CW(J),R(J),V(J),
     +                       DHAT(J)
 71         CONTINUE
            RETURN
            END

C*******************************************************************************
```

```
      SUBROUTINE INTRAN(D1,D,MAX,NLESSK)

C  SUBROUTINE TO TRANSLATE REAL DATA INTO INTEGERS...
C     (SINCE CODER/DECODER DEALS ONLY WITH INTEGERS)
C
C     D0.........INTEGER OUTPUT
C     D1.........REAL INPUT
C
C-----------------------------------------------------------------------

      INTEGER MAX,NLESSK
      INTEGER D(0:MAX-NLESSK-1)
      REAL D1(0:MAX-NLESSK-1)

      DO 1 I=0,MAX-NLESSK-1
           DO 2 J=0,MAX-1
                IF (D1(I).EQ.(J/1.0)) THEN
                    D(I)=J
                ENDIF
 2         CONTINUE
 1    CONTINUE
      RETURN
      END

C*******************************************************************************************
      SUBROUTINE BITREV(N,M,XTMP,X)

C   THIS SUBROUTINE ACCEPTS AS INPUT THE COMPLEX ARRAY
C   CONTAINING THE VALUES XTMP() WHICH WERE READ FROM
C   THE INPUT FILE.  THE OUTPUT OF THIS SUBROUTINE IS
C   THE COMPLEX ARRAY X() WHICH CONTAINS THE INPUT
C   VALUES IN 'BIT-REVERSED' ORDER.
C   (FROM THE LIBRARY OF PROFESSOR STRUM AT NPS)
C
C-----------------------------------------------------------------------

      COMPLEX XTMP(0:N-1),X(0:N-1)

      DO 10 K=0,N-1
        NEWADR=0
        MADDR=K
        DO 20 I=0,M-1
          LRMNDR=MOD(MADDR,2)
          NEWADR=NEWADR+LRMNDR*2**(M-1-I)
          MADDR=MADDR/2
20      CONTINUE
      X(NEWADR)=XTMP(K)
10    CONTINUE

      RETURN
      END


C*******************************************************************************************
      SUBROUTINE FFT(N,M,X)
```

```
C     THIS SUBROUTINE ACCEPTS AS INPUT THE COMPLEX ARRAY X(),
C     CALCULATES THE FAST FOURIER TRANSFORM (FFT) OF THE
C     ARRAY, AND RETURNS THE RESULTING SEQUENCE IN THE
C     ORIGINAL ARRAY X().
C    (FROM THE LIBRARY OF PROFESSOR STRUM AT NPS)
C
C-----------------------------------------------------------------------

      COMPLEX X(0:N-1),W,TMP
      PI=4.0*ATAN(1.0)
      EN=N

      DO 50 L=1,M
        ISPACE=2**L
        S=N/ISPACE
        IWIDTH=ISPACE/2
        DO 40 J=0,(IWIDTH-1)
          R=S*J
          ALPHA=2.0*PI*R/EN
          W=CMPLX(COS(ALPHA),-SIN(ALPHA))
          DO 30 ITOP=J,N-2,ISPACE
            IBOT=ITOP+IWIDTH
            TMP=X(IBOT)*W
            X(IBOT)=X(ITOP)-TMP
            X(ITOP)=X(ITOP)+TMP
30        CONTINUE
40      CONTINUE
50    CONTINUE

      RETURN
      END

C*********************************************************************************************************
      SUBROUTINE INVFFT(N,M,X)

C     THIS SUBROUTINE ACCEPTS AS INPUT THE COMPLEX ARRAY
C     X() AND RETURNS THE INVERSE FFT OF THE ARRAY.
C    (FROM THE LIBRARY OF PROFESSOR STRUM AT NPS)
C
C-----------------------------------------------------------------------

      COMPLEX X(0:N-1)
      EN=N

C  CALCULATE THE COMPLEX CONJUGATE OF THE INPUT DATA.

      DO 70 I=0,N-1
        X(I)=CONJG(X(I))
70    CONTINUE

C  CALCULATE THE FAST FOURIER TRANSFORM OF THE ARRAY.

      CALL FFT(N,M,X)

C  CALCULATE THE COMPLEX CONJUGATE OF THE RESULTING ARRAY.
```

```
      DO 80 I=0,N-1
        X(I)=CONJG(X(I))/EN
80    CONTINUE

      RETURN
      END
C*****************************************************************************

      SUBROUTINE GAUSS(AVG,VAR,PTS,AWGN,TEMP1,TEMP2)

C   GAUSS-SUBROUTINE TO OUTPUT A GAUSSIAN RANDOM VARIABLE ARRAY WITH
C         MEAN=AVG AND VARIANCE=VAR
C
C        AVG.......MEAN OF THE GAUSSIAN RANDOM VARIABLE
C        VAR.......VARIANCE OF THE GAUSSIAN RANDOM VARIABLE
C
C-----------------------------------------------------------------------

      INTEGER PTS
      REAL AVG,VAR,TEMP1(0:PTS-1),TEMP2(0:PTS-1)
      REAL A,B,PI
      REAL AWGN(0:PTS-1)

      PI=4.0*ATAN(1.0)
      CALL RNUN(PTS,TEMP1)
      CALL RNUN(PTS,TEMP2)
      DO 10 I=0,PTS-1
            A=-2.0*ALOG(TEMP1(I))
            B=COS(2*PI*TEMP2(I))
            AWGN(I)=SQRT(A)*B*SQRT(VAR)+AVG
 10   CONTINUE
      RETURN
      END

C*****************************************************************************
C*****************************************************************************
```

LIST OF REFERENCES

1. Paul H. Moose, "Theory of Multi-Frequency Modulation (MFM) Digital Communications," Technical Report No. NPS 62-89-019, Naval Postgraduate School, Monterey, California, May 1989.

2. Shu Lin and Daniel J. Costello, Jr., *Error Control Coding: Fundamentals and Applications,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

3. Man Young Rhee, *Error-Correcting Coding Theory,* McGraw-Hill Publishing Co., New York, 1989.

4. Richard E. Blahut, *Theory and Practice of Error Control Codes,* Addison-Wesley Publishing Co., Inc., Reading, Massachusetts 1983.

5. Huijie Deng and Daniel J. Costello, Jr., "Fast Decoding of d(MIN) = 6 RS Code," Technical Report No. NASA-CR-173620, Illinois Institute of Technology, Chicago, Illinois, December 1983.

6. Daniel J. Costello, Jr., Huijie Deng, and Shu Lin, "Fast Decoding Techniques for Extended Single-and-Double-Error-Correcting Reed Solomon Codes," Technical Report No. NASA-CR-173617, Notre Dame University, Notre Dame, Indiana, January 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor P.H. Moose, Code EC/Me Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	6
5.	Capt. T.A. Schwendter, U.S.A.F. Code EC/Sc Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Commander Naval Ocean Systems Center Attn: Mr. Darrell Marsh (Code 624) San Diego, CA 92151	3
7.	Commanding Officer Naval Submarine School Attn: Lt. Robert W. Ives, U.S.N. (SOAC Class cvn. 16JUL90) Box 700 Groton, CT 06349-5700	5